Old Dominion University Research Foundation

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

BUILDING A GENERALIZED DISTRIBUTED SYSTEM MODEL

By

R. Mukkamala, Principal Investigator

P-124

Progress Report
For the period February 1, 1992 to July 31, 1992

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665

Under
Research Grant NAG-1-1114
Wayne H. Bryant, Technical Monitor
ISD-Systems Architecture Branch

July 1992

1 Introduction

In the 1991-92 proposal, we presented a scheme to implement the distributed system simulator. To this end, we have designed and implemented the modules. These were written in C. The graphics interface to the system is yet to be developed. In the coming year, we propose to complete the graphics interface and use the prototype system to validate some theoretical models of prototypes.

The PhD student supported by this grant has successfully completed the candidacy examination. The student's plan of research for the thesis has also been approved by the thesis committee. The research is now in progress. The proposal reports are enclosed here.

We have also looked at storage efficient schemes to implement replicated distributed systems. The results from this research are accepted for publication in the IEEE Transactions on Knowledge and Data Engineering. In addition, we have also looked at the modeling of gracefully degrading computing systems. These results will appear in Microelectronics and Reliability Journal.

In this report, we summarize our progress in these four areas and then describe the proposed work for 1992-93.

2 Distributed System Prototype

The work has centered on building a distributed system prototype that would help us study the performance determinants, bottlenecks, and the effect of various algorithms and policies on the system. We have designed and built a flexible distributed system prototype which is very general in the sense that modification of a module or segment of the code to test a new policy or to add a new policy can be done with ease without concern about reconfiguring the whole system.

The system comprises of independent modules that communicate among each other to achieve an objective which in our case is the processing of a transaction. We define a *transaction* to be an atomic processing entity that transforms the stored data from one consistent state to another. The stored data in question are the data objects (or resources) that can be manipulated by using such operations as *read*, *write* and *compute*. A typical transaction is a sequence of reads and writes (the introduction of the operation of compute is trivial and does not help us study anything new). Following is a

brief description of the various modules involved and their functionalities including the various policy options governing their operation.

2.1 System Description

The various modules that are involved and their interactions are shown in Figure 1. Currently, sites (or nodes) are assumed to be identical in behavior with respect to transaction processing. In other words, heterogeneous handling of transactions is not introduced into the system at this stage. The figure also shows the operations that involve cooperation between the system modules to execute distributed transactions. The attached table (see Appendix) gives a complete listing of the op-codes and the corresponding actions.

The functionalities of the modules can be best appreciated by viewing the stages of processing a transaction goes through. The user at a working terminal seeks to run a transaction which involves reads and writes of data objects that reside at possibly multiple sites. The transaction is submitted to the *User Transaction Manager* or UT which verifies the syntax of the transaction and transfers it to the *Global Transaction Manager* (GTM). The function of the GTM is to keep track of the state of all active transactions both remote and locally processed ones. GTM parses the query and finds the required objects and the operations to be performed on them. Each such entity (the object-operation pair) is treated as a *subtransaction*. The execution of the transaction now involves the the execution of these subtransactions. Now the GTM requests the *Replica Controller* (RC) for the location and the quorum (for read/write) information about each object. The replica controller maintains a list of all the sites participating in the functioning of the overall system, information about the objects residing at the site viz., object-id, site-id, votes required for read and write.

Many replica control algorithms were proposed. Among them, vote assignment and *coterie* are two of the best. In the method of vote assignment, a number of votes are assigned to each site in the system, and a group whose members have a majority of the total votes (called majority group) is allowed to performed the operations. Mutual exclusion is achieved because at a given time at most a single group can have a majority of votes at a time. The disadvantage of this method is the operation can not performed if no group has a majority. Lamport suggested the method of *coterie*. In this method, we define a set of groups that may perform the operations. Each pair of groups should have a node in common to guarantee mutual exclusion.

The policy we have a chosen is a special case of the first policy in which, the size of the majority group is one for *read* operations and equal to the total number of sites holding copies of the data object, for *write* operations. We call it the *read one write all* policy which means that to perform a read operation on a data object one needs to acquire a lock (shared) from any one of the sites at which the data object resides, however to perform a write operation we need to get locks (exclusive) from all sites at which there is a copy of the object in question.

Based on the response from RC, the GTM sends requests to the *Global Concurrency Control Manager* (GCCM) to acquire the locks for the objects to be manipulated. At this stage objects could be local to the site or residing on a remote site. If an object resides at a remote site then the lock request command is passed to the corresponding sites GCCM. The quorum number is used to check if the required number of sites have responded (granted locks) so that the further processing can proceed. The GCCM sends local lock requests to the *Local Concurrency Control Manager* (LCCM). Depending upon the availability of the object at the time the request is made (some other transaction could have gained an exclusive lock) a lock granted or a lock refused message is sent back by the LCCM.

The GCCM upon receiving the required number of locks granted messages sends a response to the GTM indicating that the quorum is satisfied. The GTM now sends messages that involve the execution of each subtransaction. Subtransactions involving objects residing at remote sites are passed to the GTM at the corresponding remote site, after which its processing follows the same lines as the normal (local) subtransaction. Subtransactions that are local (also include subtransactions received by the GTM from other sites) are passed to the *Local Transaction Manager* (LTM). At this stage there is no way to distinguish between a local and remote transaction. Subtransactions may involve either reading or writing (modifying) an object.

If its a *read* operation then the LTM directly contacts the *Resource Manager* (RM) to perform the read operation. The RM acts as a scheduler for read write requests for read write requests. The object in question is read and its value is returned as a response to LTM. If however, the operation is a *write* operation, the LTM sends a "logical write request"(different from physical write - to implement *two-phase commit*) to the *Local Transaction Recovery Manager* (LTRM). The LTRM maintains the *commit* status of subtransactions. The LTRM on receiving a logical write request stores the object-id and the corresponding value in a data structure (physical write

is still not done). Also a logical write done response is sent to the LTM, which is propagated by the LTM to the GTM. On receiving/not receiving successful write done(logical) from all (within the quorum limit) sites, the GTM makes a decision to commit/abort the subtransaction. A *two-phase commit* protocol is used to ensure that either all sites commit or all abort a transaction thus maintaining data consistency.

2.2 Current Status

The prototype has been built and is being currently tested. Currently we are looking for bottlenecks in the system. We are looking for the possibility of deadlocks and the mechanisms one can use to avoid/prevent them(and obviously the cost involved in doing do). The impact of policies is another important phase of the project, we are currently investigating upon the policies that might have to be considered in the various modules of the system. With multiple user transactions being executed concurrently and depending upon the policies chosen in each module we might face problems that are typical of the policy or the combination of them (e.g., Read Write hazards). We intend to develop an interactive, menu driven graphical user-interface, once we introduce a more concrete and practical concept of the user-transaction. Work towards this end is almost complete, however it is not yet implemented.

3 Distributed Real-time Systems: Thesis work

Due to the importance of reliability and timeliness in real-time systems, the application of distributed systems in this area is now well recognized. In this context, we propose to look at the issues of mutual exclusion and replica control in these systems. A complete summary of the proposed work is enclosed with this report. Here, we summarize the proposed work.

In the context of mutual exclusion problems, we propose to investigate the following issues.

- **Develop criteria to classify/evaluate mutual exclusion algorithms of distributed real-time systems.** This work should result in metrics to express the suitability of a given ME algorithm to distributed real-time applications. Even though the optimistic algorithms appear to be more suitable to real-time applications than the conservative ones, it is not clear if this classification is sufficient.

- **Suggest modifications to existing mutual exclusion algorithms to meet the needs of real-time applications.** Having arrived at a classification, we propose to analyze some of the existing ME algorithms and classify them accordingly. In addition, we propose to suggest modifications to the algorithms to transfer them from a less suitable class to a more desirable class. This should be possible by changing the grant/release rules in an algorithm.

- **Develop new mutual exclusion algorithms for distributed real-time systems.** Using the properties derived from the classification, we will attempt to construct new algorithms that are suitable for real-time applications. In fact, it may result in a suite of algorithms where the choice will depend on the semantics of the application.

- **Evaluate the algorithms using the criteria developed above.** This may involve using both analytical and simulation tools.

- **Develop guidelines for future development.** If in the process of development and analysis, we have developed sufficient insight regarding the applications and the algorithms, we may be able to develop some general guidelines for future work. However, this should be perceived more as wishful thinking than as a promised delivery.

In the context of replica control, we propose to investigate the following issues.

- Model some typical application related semantics where replication is needed to improve the reliability and availability.

- Develop criteria to evaluate the replica control algorithms of distributed real-time systems.

- Study the existing replica control algorithms in the environment of real-time applications. Characterize them in terms of their suitability to real-time systems.

- Develop new replica control algorithms for distributed real-time systems.

- Evaluate the algorithms with the established criteria.

The plan of research is also summarized in the enclosed proposal document.

4 Measuring the Effects of node clustering on system performance

This research deals with the effects of node clustering on a distributed token-ring based scheme in a distributed system. While almost all the existing schemes treat each of the physical site as a node in the logical ring, it creates performance bottlenecks due to the long propagation times along the physical ring. This is especially inefficient when the requests for a resource are nonuniform among the nodes. We propose and analyze the performance of a mutual exclusion scheme based on such clustering. We conclude that

- Token-ring based mutual exclusion algorithms are better suited under heterogeneous loads than homogeneous loads.
- Clustering nodes based on their load patterns significantly improves system performance.
- The behavior of the clustered system is more complex to analyze than an unclustered system.

The results of this work are presented at the Pittsburgh Modeling and Simulation Conference, May, 1992. The complete paper is enclosed here.

5 Storage efficient and secure replicated distributed databases

Data availability and security are two important issues in a distributed database system. Existing schemes achieve high availability at the expense of higher storage cost, and data security at the expense of higher processing cost. In this work, we develop an integrated methodology which combines the features of some existing schemes dealing with data fragmentation, data encoding, partial replication, and quorum consensus concepts to achieve storage efficient, highly-available, and secure distributed database systems. The results from this work will appear in IEEE Transactions on Knowledge and Data Engineering in 1992. A complete copy of the paper is enclosed here.

6 Modeling and analysis of Gracefully Degrading Computing Systems

Much of the existing work on gracefully degrading computing systems are only applicable when the life distribution for the system components is exponential. In this work, we develop an order-statistics based model for these systems. We show the generality of this model by deriving expressions for system reliability, mean-time-to-failure, and mean-computations-before-failure. In addition, we derive some interesting properties for the failure rate of such a system . In particular we show that the failure rate of a gracefully degrading system with i.i.d. DFR (Decreasing Failure Rate) components is also DFR if the coverage probability is less than 1/2. This generalizes a well known result for series systems. The results from this work will appear in Microelectronics and Reliability Journal in 1992. A complete copy of the paper is enclosed here.

7 Summary of Accomplishments in 1991-92

We have published the results of our research (since August 1991) in one conference proceeding and two journals. Currently, we are working on two journal papers. In addition, the PhD student funded by this project has written two reports related to the distributed real-time systems. These form the basis for the future PhD thesis work.

1. Y. Kuang and R. Mukkamala, "Measuring the effects of node clustering on the performance of token ring based mutual exclusion algorithms," *1992 Modeling and Simulation Conference*, Pittsburgh, PA, May 1992.

2. N.R. Chaganty and R. Mukkamala, "Order Statistics based modeling of gracefully degrading computing systems," To appear in *Microelectronics and Reliability Journal*, 1992.

3. R. Mukkamala, "Storage efficient and secure replicated distributed databases," TO appear in *IEEE Trans. Knowledge and Data Engineering*, 1992.

4. Y. Kuang, "Design and analysis of distributed real-time systems," A report prepared for the PhD thesis committee, 1992.

5. Y. Kuang, "Mutual exclusion and replica control in distributed real-time systems: A proposal," 1992.

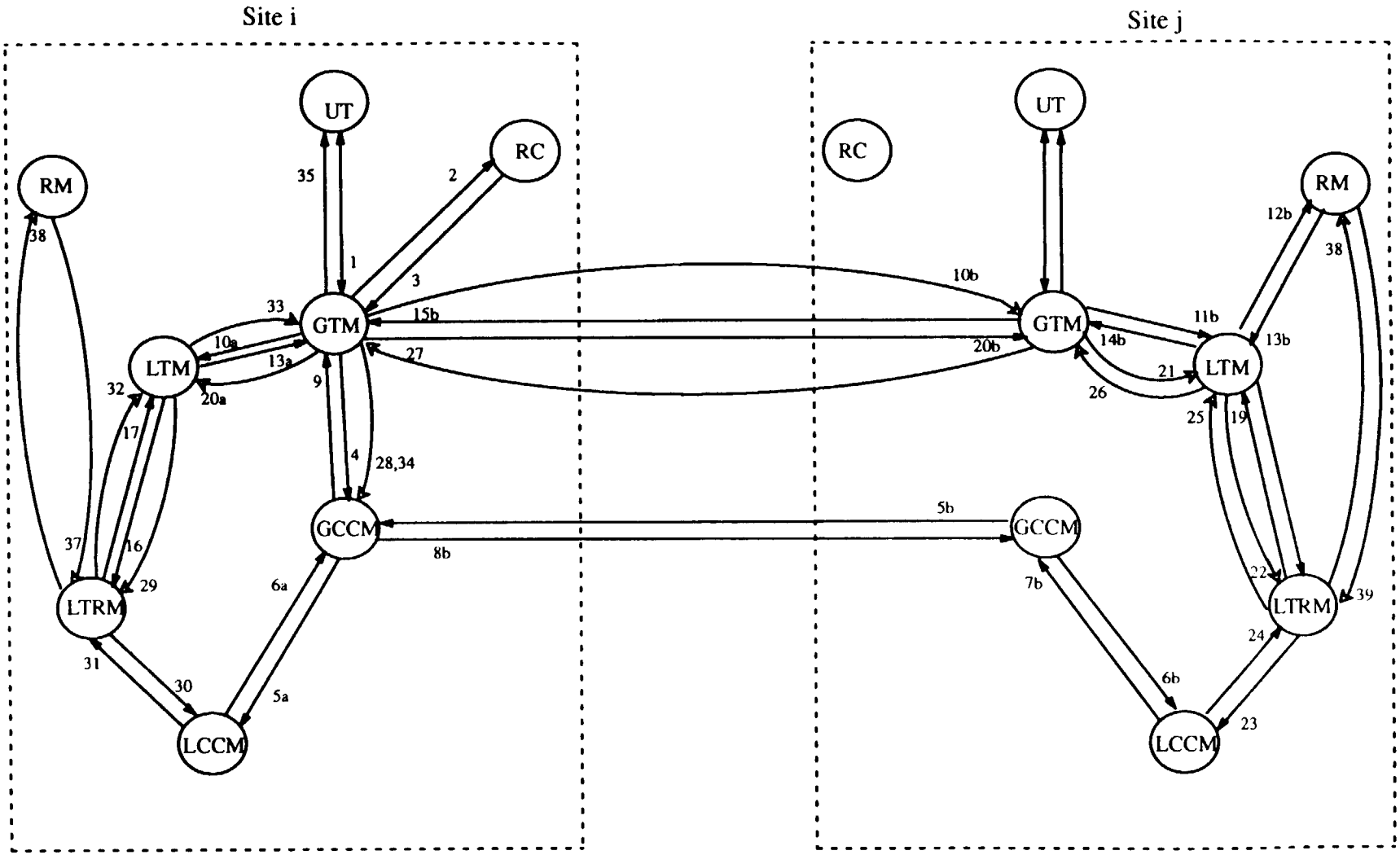
In addition, our current work on building the prototype for a distributed system should result in several conference papers in 1992-92.

8 Proposed Research Efforts in 1992-93

During the next grant period (August 1992 - July 1993), we propose to continue the development and testing of the distributed prototyping system. We also wish to employ it in determining the performance of selected schemes in distributes systems. The main problems we propose to solve during this period are:

- Complete the testing of the prototype.
- Enhance the functionality of the modules by enable the experimentation with more complex protocols.
- Use the prototype to verify the theoretically predicted performance of locking protocols, etc.
- Work on issues related to real-time distributed systems. This should result in efficient protocols for these systems.

Data Control Flow Diagram for the Distributed System Prototype



APPENDIX

Distributed System Prototype: Operational Description

Notation	Description
GCCM	Global Concurrency Manager
GTM	GLobal Transaction Manager
LCCM	Local Concurrency Manager
LTM	Local Transaction Manager
LTRM	Local Transaction Recovery Manager
RM	Recovery Manager
RMC	Replica Controller
UTM	User Transaction manager

Op_code#	Operation Explanation(with format)
1	UT module sends a transaction to GTM. The transaction may look like: Read A Read B Write C Write D where each Read/Write is called a "Subtransaction". The message format is : [USER_TRANSACTION_REQUEST(L)][USER_TRANSACTION_BEGIN(I)] [USER_ID(I)][READ_OP(I)][ITEM_ID(I)][WRITE_OP(I)] [ITEM_ID(I)][DATA(I)]...[USER_TRANSACTION_END(I)]
2	GTM makes subtarnsactions for every Read/Write of an item (eg. A, B) and sends a request to RC for complete knowledge of replication and qourum needed for R/W. The message format is : [QOURUM_READ(WRITE)_REQUEST(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)]
3	RC finds an optimal list of sites needed for R/W qourum of an item in a subtransaction. RC sends this list to GTM. The message format is :
a.	[QOURUM_READ(WRITE)_REPLY(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][QUORUM(I)][NUM_SITES(I)] [SITENAME1(I,S)][VOTE1(I)]...
b.	[QOURUM_READ(WRITE)_REFUSED(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)]

Op_code#	Operation Explanation(with format)
4	For Subtx GTM passes the list of sites obtained from RC to GCCM for making lock requests to local/remote site LCCM. The message format is : [LOCKS_REQUEST(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][QUORUM(I)][NUM_SITES(I)] [SITENAME1(I,S)][VOTE1(I)]...
5a	If the resource to be R/W for a subtx is local, a lock request is made to LCCM by GCCM. The message format is : [LOCK_READ(WRITE)_REQUEST_LOCAL)(L)][USER_TRANSACTION_ID [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)]
5b	If the item is at a remote site, the request is transferred such that GCCM at the remote site handles it as if the request came from remote site's GTM. The message format is : [LOCK_REQUEST_REMOTE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][QUORUM(I)][NUM_SITES(I=1)] [SITENAME1(I,S)][VOTE1(I)]
6b	At remote site, above recieved request is transferred as mentioned in step 5a. The message format is : [LOCK_READ(WRITE)_REQUEST_LOCAL)(L)][USER_TRANSACTION_ID [SUB_ID(I)][ITEM_ID(I)]

Op_code#	Operation Explanation(with format)
6a	LCCM replies GCCM for a lock request with lock granted/refused messages. GCCM keeps track of all such granted/refused messages and #votes associated with them The message format is :
a.	[LOCK_READ(WRITE)_GRANTED_LOCAL)(L)] [USER_TRANSACTION_ID(I)][SUB_ID(I)][ITEM_ID(I)] [VERSION(I)]
b.	[LOCK_READ(WRITE)_REFUSED_LOCAL)(L)] [USER_TRANSACTION_ID(I)][SUB_ID(I)][ITEM_ID(I)]
7b	see 6a.
8b	Remote GCCM replies to the GCCM where the lock request originated with granted/refused reply. Also #votes associated are passed. The message format is :
a.	[LOCK_GRANTED_REMOTE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][VERSION][SITENAME(S)][VOTE(I)]
b.	[LOCK_REFUSED_REMOTE)(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][SITENAME(S)][VOTE(I)=0]
9	When GCCM has :
a.	received enough votes to beat the R/W quorum for a subtx then a subtx's all lock granted reply is sent to the GTM. The message format is : [LOCKS_GRANTED(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)][QUORUM(I)][NUM_SITES(I)] [SITENAME1(I,S)][VERSION(I)]...
b.	Failed to receive enough votes for a subtx, then an "abort" is passed to GTM. The message format is : [LOCKS_REFUSED(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][R/W_ID(I)]

Op_code#	Operation Explanation(with format)
35	
a.	If GTM gets a "lock abort" for a subtx of an incomplete Tx, it sends a abort to UT. The message format is : [USER_T_ABORTED(L)][USER_TRANSACTION_ID(I)]
b.	Send a "abort" to all LTMs where a lock is held. If a local lock is held send "abort" to local LTM, else if lock is held at a remote site, send "abort" to remote GTM
20a	Local message format is : [TRANSACTION_ABORT_LOCAL(L)][USER_TRANSACTION_ID(I)]
20b	Remote message format is : [TRANSACTION_ABORT_REMOTE(L)][USER_TRANSACTION_ID(I)]
10a	
a.	If a subTx is a READ(local): A "read" operation is sent to LTM. The message format is : [READ_REQUEST_LOCAL(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]
b.	If a subTx is a WRITE(local): A "write" operation is sent to LTM. The message format is : [WRITE_REQUEST_LOCAL(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][VERSION(I)][DATA(I)]

Op_code#	Operation Explanation(with format)
10b a.	If a subTx is a READ(remote): A "read" operation is sent to remote GTM. The message format is : [READ_REQUEST_REMOTE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]
b.	If a subTx is a WRITE(remote): A "write" operation is sent to remote GTM. The message format is : [WRITE_REQUEST_REMOTE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][VERSION(I)][DATA(I)]
11b	same as 10a (Now request is local).
11a,12b	Local READ operation: LTM passes the read operation to RM of the site. The message format is : [PHYSICAL_READ_REQUEST(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]
12a,13b	Local READ reply from RM to LTM. The message format is : [PHYSICAL_READ_DONE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]
13a,14b	Local READ done reply from LTM to GTM: The message format is : [READ_DONE_LOCAL(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]

Op_code#	Operation Explanation(with format)
15b	Remote READ done is passed back to the GTM at which it originated. The message format is : [READ_DONE_REMOTE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]
10a,11b	Local WRITE operation: GTM passes the write operation for an item mentioned in a subTx to its LTM. The message format is : [WRITE_REQUEST_LOCAL(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]
16,18	LTM passes WRITE operation to LTRM : The message format is : [LOGICAL_WRITE_REQUEST(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]
17,19	LTRM sends "write done" reply to LTM. LTRM actually stores the value in a datastructure. (Physical write is still not done.) The message format is : [LOGICAL_WRITE_DONE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]
13a,14b	A "prepared" message is sent to GTM by LTM when a "write done reply" is received. The message format is : [PREPARED(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]

Op_code#	Operation Explanation(with format)
20a,21	When correct # of "prepared" messages are/(are not) received by GTM for all write subTx's of a Tx, a commit/abort message is sent to local LTM for updates(final physical WR) and all GTMs(remote) which are involved in updates. local message formats are(commit and abort)(GTM to LTM): [TRANSACTION_COMMIT_LOCAL(L)][USER_TRANSACTION_ID(I)] [TRANSACTION_ABORT_LOCAL(L)][USER_TRANSACTION_ID(I)]
20b	Remote message formats are(commit and abort)(GTM to GTM): [TRANSACTION_COMMIT_REMOTE(L)][USER_TRANSACTION_ID(I)] [TRANSACTION_ABORT_REMOTE(L)][USER_TRANSACTION_ID(I)]
22,29	Commit or Abort is passed to LTRM by LTM The message format is : [COMMIT_LOCAL(L)][USER_TRANSACTION_ID(I)] [ABORT_LOCAL(L)][USER_TRANSACTION_ID(I)]
23,30	Lock release requests for every write/read lock held for that Tx is sent to LCCM The message format is : [LOCK_RELEASE_REQUEST(L)][COMMIT(ABORT)] [USER_TRANSACTION_ID(I)]

Op_code#	Operation Explanation(with format)
31,24	Lock released is acknowledged by LCCM: The message format is : [LOCK_RELEASE_DONE(L)][USER_TRANSACTION_ID(I)]
36,38	"Physical writes" are sent to the RM. The message format is : [PHYSICAL_WRITE_REQUEST(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)][DATA(I)]
37,39	The "Physical write done" is sent back LTRM. The message format is : [PHYSICAL_WRITE_DONE(L)][USER_TRANSACTION_ID(I)] [SUB_ID(I)][ITEM_ID(I)]
32,25	"Commit Done" is sent back to LTM. The message format is : [COMMIT_DONE(L)][USER_TRANSACTION_ID(I)]
33,26	"Commit Ack" is sent back to GTM. The message format is : [COMMIT_ACK(L)][USER_TRANSACTION_ID(I)]
27	Remote GTM sends the "Commit Ack" back to original GTM The message format is : [COMMIT_ACK_REMOTE(L)][USER_TRANSACTION_ID(I)]

Op_code#	Operation Explanation(with format)
35	GTM when receives enough # of commits ACKs from all involved sites, it announces "User Transaction Done" to UT The message format is : [USER_TRANSACTION_DONE(L)][USER_TRANSACTION_ID(I)]
28	Kill transaction from GTM to GCCM (Abort sent by GTM to LTM) The message format is : [KILL_TRANSACTION(L)][USER_TRANSACTION_ID(I)]
34	"Done Transaction" from GTM to GCCM (Commit sent by GTM to LTTM) The message format is : [DONE_TRANSACTION(L)][USER_TRANSACTION_ID(I)] GCCM should release all datastructures for that Tx.
38	"Read_Value" from GTM to UT For every subtransaction, when a read_done comes from LTM or remote GTM, this message with a return value is passed to UT process that generated it. The message format is : [READ_VALUE(L)][USER_TRANSACTION_ID(I)][SUB_ID(I)] [ITEM_ID(I)][VALUE(I)]

MEASURING THE EFFECTS OF NODE CLUSTERING ON THE PERFORMANCE OF TOKEN RING BASED MUTUAL EXCLUSION ALGORITHMS

Yinghong Kuang Ravi Mukkamala

Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

ABSTRACT

Token-ring based mutual exclusion algorithms are attractive due to their operational simplicity and the guaranteed starvation-free and deadlock-free operations. Existing studies have looked at the behavior of these algorithms under homogeneous load conditions. In this paper, we investigate their behavior under heterogeneous loads. In addition, to reduce the long idle token rotations, we suggest clustering of nodes. The behavior of the system under clustering is also investigated. We conclude that the token-based mutual exclusion algorithms display improved performance under load heterogeneity and node clustering.

1. INTRODUCTION

A distributed system consists of a set of processing nodes connected through a communication subsystem. The nodes have no shared memory and communicate with one another through message passing. The system also contains a set of resources shared among the nodes in the system. Providing mutually exclusive access to the shared resources is a fundamental responsibility of a distributed system.

A number of distributed algorithms which implement system-wide mutual exclusion have appeared in the literature [1, 3, 5, 8, 9]. These can be classified into two classes: token-based and non-token based. In token-based algorithms, only the node holding the token can enter the critical section to access the resource in a mutually exclusive fashion. In general, the current token holder decides on the next node to whom the token should be passed to. In a non-token based algorithm, a node seeking mutually exclusive access should gain permission from either all or a subset of nodes in the system prior to entering the critical section. In general, the non-token based algorithms involve more messages per request than token-based systems.

In this paper, we are considering a special set of token-based algorithms: token-ring based algorithms. Here, nodes in a distributed system are connected in a *logical* ring structure. A special message called *token* rotates around the ring. A node can access a shared resource only when it receives a token, which is handed to it by its logical neighbor on the ring. The fact that the system incurs a fixed overhead of rotating the token along the ring continuously has deterred many distributed systems from using this mechanism to implement mutual exclusion. However, when reasonable load is present on the shared resource, then the fixed overhead in a token ring more than compensates for the multiple messages that are exchanged in other token based systems and especially in non-token based systems.

Even though a number of performance analyses have been carried out on token-ring based systems, almost all the studies assume uniform load among the nodes in the system [2, 6]. Even though such assumptions are attractive for analytical tractability, they do not represent in practical situations. For this reason, in this paper, we concentrate on heterogeneous loads. Analysis of systems under homogeneous loads is only used as a means of comparison for heterogeneous loads. In addition, to reduce the idle token rotation time along the ring, we introduce the concept of node clustering. Here, nodes are grouped into clusters, and the clusters are connected through a logical ring, thereby reducing the total number of entities on the ring. We also determine the impact of clustering on the performance of the system. Unlike several studies that assume exponential arrival of requests at each node, we assume a process model at each node. The model is explained in detail in Section 2.

2. MODEL

We model the distributed system as a set of n autonomous nodes connected through a communication system. Each node runs a single process executing an infinite loop. Each process loop consists of three stages: (i) Compute, (ii) Wait for a resource, and (iii) Access the resource by entering the CS (critical section) and then release the resource. This is described in Figure A. We assume that there is only one shared resource in the system.

The time process P_i stays in the compute state in any iteration is assumed to be exponentially distributed with mean c_i (sec.). Similarly the time that a process retains the resource (or remains in CS) is assumed to be exponentially distributed with mean s_i (sec.). The distribution of the time in the wait state (w_i) depends on the token propagation policy and the requirements of other nodes in the system.

When we consider a homogeneous system, all n nodes have the same characteristic. Hence, $c_1 = c_2 = \ldots = c_n = c$ and $s_1 = s_2 = \ldots = s_n = s$. We express the load offered by the n nodes on the shared resource as $n \cdot s/c$ or $100 \cdot n \cdot s/c$ when expressed as a percentage.

In the case of heterogeneous systems, we consider two types of nodes: high-load nodes and low-load nodes. Assuming that the average service time is the same for both nodes, we distinguish the two types of nodes by the average time in the compute state: c_l and c_h.

Node Clustering:

In addition to determining the effect of load heterogeneity, we are also interested in finding the effects of node clustering on mutual exclusion algorithms. Especially, when practical distributed systems are built as clusters of nodes, this aspect is extremely important.

Based on the physical proximity, nodes are clustered. For simplicity, we assume clusters of equal size. For example, when 100 nodes are grouped into 10 clusters, each cluster will have 10 nodes. In addition, each cluster will offer approximately the same load on the shared resource. Thus, load heterogeneities at the node level do not appear at the cluster level. (We are currently looking into the load heterogeneity at the cluster level.) The clusters are connected through a logical token ring. Hence, only adjacent clusters communicate. The nodes within a cluster, however, have point-to-point connections. Hence, any node can communicate to any other node within its cluster. One of the nodes in the cluster acts as an agent for the cluster in the logical ring. This agent is responsible for managing the token on behalf of the cluster: receiving the token from the preceding cluster, distributing the token within the cluster, and delivering it to the successor cluster on the ring.

Mutual Exclusion Protocol:

Since we are dealing with mutual exclusion in a clustered system, we need to define the protocol adopted within a cluster and among the clusters. Following protocol is adopted in our study.

Step 1. Within a cluster, nodes contend for mutual exclusion using a typical non-token based algorithm such as Lamport's algorithm [3] or Ricart and Agrwala's algorithm [8]. The message delay between any two nodes within a cluster is assumed to be the same. A node that has obtained permission from all other nodes cannot access the resource (or enter CS)

until its cluster agent has obtained the token.

Step 2. When a cluster agent receives a token, it determines if any of its nodes require it. If no node requires it, then the token is passed to the next cluster. Otherwise, the token is passed to the node to whom the agent has given prior permission to enter CS.

Step 3. When a node within a cluster releases the resource, it passes the token to the next contender within the cluster. This process continues until no more nodes require the resource. At this time, the agent passes the token to the next cluster.

In this paper, we are not considering issues such as node failures, agent failures, or link failures. We assume the use of existing distributed election mechanisms to handle such events [7].

3. RESULTS

Figures 1-3 summarize the results obtained in this study. Since analysis is almost intractable, we have used simulation to obtain these results. Each figure contains results obtained with uniform and heterogeneous loads. (Note: In all the figures, the left-hand side graphs correspond to the heterogeneous case and the right-hand side to the uniform case.) In the case of heterogeneous load, we assume a 20/80 policy which implies that 20% of the nodes have 80% of the overall load requirement for the shared resource. In other words, out of the 100 nodes that we have considered, 20 nodes are high-load type and contribute to 80% of the requests for the resource. The remaining 80 nodes are low-load type and contribute to 20% of the requests for the resource. The simulation is carried out with two types of clustering: (i) Group the 100 nodes into ten clusters, each containing ten nodes. (ii) Group the nodes into twenty clusters each containing five nodes. In each case, we have ensured that the total demand for the shared resource is equally distributed among the clusters. In addition, we have considered the unclustered case with 100 nodes.

Since the system behavior is dependent on the propagation delay (or idle token rotation time) along the ring and the average service time (s) at the shared resource (per request), we have experimented with ratios of these two times. If d is the average propagation time between any two adjacent nodes on the ring, and c is the number of clusters, then $c \cdot d$ is the idle token rotation time. We kept the average service time constant at 1 second, and varied d. In particular, we have experimented with values of $d = 0.01, 0.1, 1.0$ (in sec.).

We have considered several load factors: low loads such as 10% and 25%, medium loads of 50% and 75%, and high loads of 100%.

Figure 1 illustrates the effects of the load factors, load heterogeneity, and node clustering on average response time. Here, response time is measured as the time elapsed between when a node desires to acquire mutually exclusive access to the resource to the time it releases the resource. Hence, it includes both the waiting time and the service time.

Figure 2 summarizes the average resource utilization under different conditions. Here, resource utilization refers to the fraction of the time the shared resource is used by any of the nodes. Clearly, the utilization of the resource is limited by the load factor. However, other factors, such as idle token rotation time and load distributions may substantially reduce the utilization.

Figure 3 summarizes the average token rotation times under different conditions. In each token rotation time, $c \cdot d$ sec. is the fixed component due to idle propagation time around the ring. The rest is attributed to service times at the nodes. Thus, increase in average token rotation time indicates increase number of services per token rotation.

We will now attempt to explain the observed behavior of the system, and draw some valid conclusions.

First, let us consider the effects of load heterogeneity with no clustering of nodes. Here, the idle token rotation time is $100d$ sec. which varies from 1 sec. when $d = 0.01$ to 100 sec. when $d = 1.0$. It may be observed that

- The response time is significantly improved with load heterogeneities. One of the prime reasons for this improvement is the reduction in synchronous-like behavior of token usage observed in token-rings with uniform loads [4]. Under uniform loads, there is a tendency to form two kinds of token rotations: one in which no node uses the token and the other in which many attempt to use it. In other words, the usage of the resource is not uniform over time under uniform load. However, under the heterogeneous load, the idle token rotations are reduced considerably, resulting in less variations in token rotation times. This improves the average response time at the nodes. This is an important result.

- The resource utilization is slightly lower with heterogeneous loads. This is especially noticeable when $d/s = 0.1$. To explain this phenomenon, we need to examine the underlying node model. Since each node represents a process with a compute, wait for the token, access and resource release cycle, the requests for the resource depend on if earlier request was fulfilled. In the case of heterogeneous load, the high-load nodes have smaller computation times. However, once the token is released, even if it has finished its computation, a node has to wait until the token comes around. In the meanwhile, the token has visited other low-nodes with a low probability of usage. Hence, the reduced resource utilization. This observation may not be valid in a typical system with independent requests arriving at a node.

- The average token rotation is significantly lower with heterogeneous load. This confirms the above observation of lower utilization as well as the fact that the usage is better distributed among the token rotations than in a uniform load condition.

Now let us consider the effect of clustering on system performance. The main effect of clustering is to reduce the idle token rotation time. The effect is more significant when $d = 1.0$ where the idle rotation time is reduced from 100 sec. to 20 sec. in a 20-cluster system and 10 sec. in a 10-cluster system. This is a significant savings. The effect is less visible when $d = 0.01$ where the 1 sec. rotation time is reduced to 0.2 sec. in a 20-cluster system and 0.1 sec. in a 10-cluster system. The second effect of clustering is to reduce idle token rotations. Since a cluster has a number of nodes the probability of at least one node requiring to access the resource when a token is received by the cluster is much higher than in an unclustered configuration. This significantly improves the resource utilization. In summary,

- Clustering improves response time under heterogeneous and uniform loads. The improvement is attributed is less variations in token rotation times due to better distribution of requests among the token rotations. For this reason, the improvement under heterogeneous load is more distinct than under uniform load.

- Clustering has no impact on the overall utilization under homogeneous loads. Its impact under heterogeneous loads is quite complex to explain. While the utilization is significantly reduced in a 10-cluster system at $d = 0.01$, the 20-cluster system has better utilization for higher values of d. Further investigations are required to explain this behavior.

- Under homogeneous load, clustering has significantly higher token rotation time than the unclustered. This is not an obvious result. While the idle token rotation time has reduced with clustering, the services per visit of a token have increased. This has resulted in an increase of the token rotation time. Under heterogeneous load, the token rotation behavior is similar to that of utilization as explained above.

In summary, we conclude that:

- Token-ring based mutual exclusion algorithms are better suited under heterogeneous loads than homogeneous loads.

- Clustering nodes based on their load patterns significantly improves system performance.

- The behavior of the clustered system is more complex to analyze than an unclustered system.

4. CONCLUSION

In this paper, we have proposed and analyzed the performance of token-ring based distributed mutual exclusion algorithms. Especially, we looked at the effects of heterogeneous load and node clustering on system performance. The system performance has significantly improved under heterogeneous loads. Clustering has also improved the performance. We are currently studying algorithms to manage multiple resources based on the same principles as explained in this paper. These are relevant in transaction environments where a transaction may require one or more resources (i.e. data items) prior to start of execution.

5. ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contract NAG-1-1114.

References

[1] Y. I. Chang, M. Singhal, and M. T. Liu, "A dynamic token-based distributed mutual exclusion algorithm," *IEEE Phoenix Conf. Computers and Communications*, pp. 240-246, 1991.

[2] A. Gravey and A. Dupuis, "Performance evaluation of two mutual exclusion distributed protocols via Markovian modeling," *Proc. Sixth IFIP Workshop on Protocol Specification, Testing, and Verification*, pp. 335-346, 1987.

[3] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, Vol. 21, No. 7, pp. 558-565, 1978.

[4] R. J. T. Morris and Y. T. Wang, "Some results for multi-queue systems with multiple cyclic servers," *Performance of Computer Communication Systems, IFIP*, pp. 245-258, 1984.

[5] S. Nishio, K. F. Li, and E. G. manning, "A resilient mutual exclusion algorithm for computer networks," *IEEE Trans. Parallel and Dist. Systems*, Vol. 1, No. 3, pp. 344-355, July 1990.

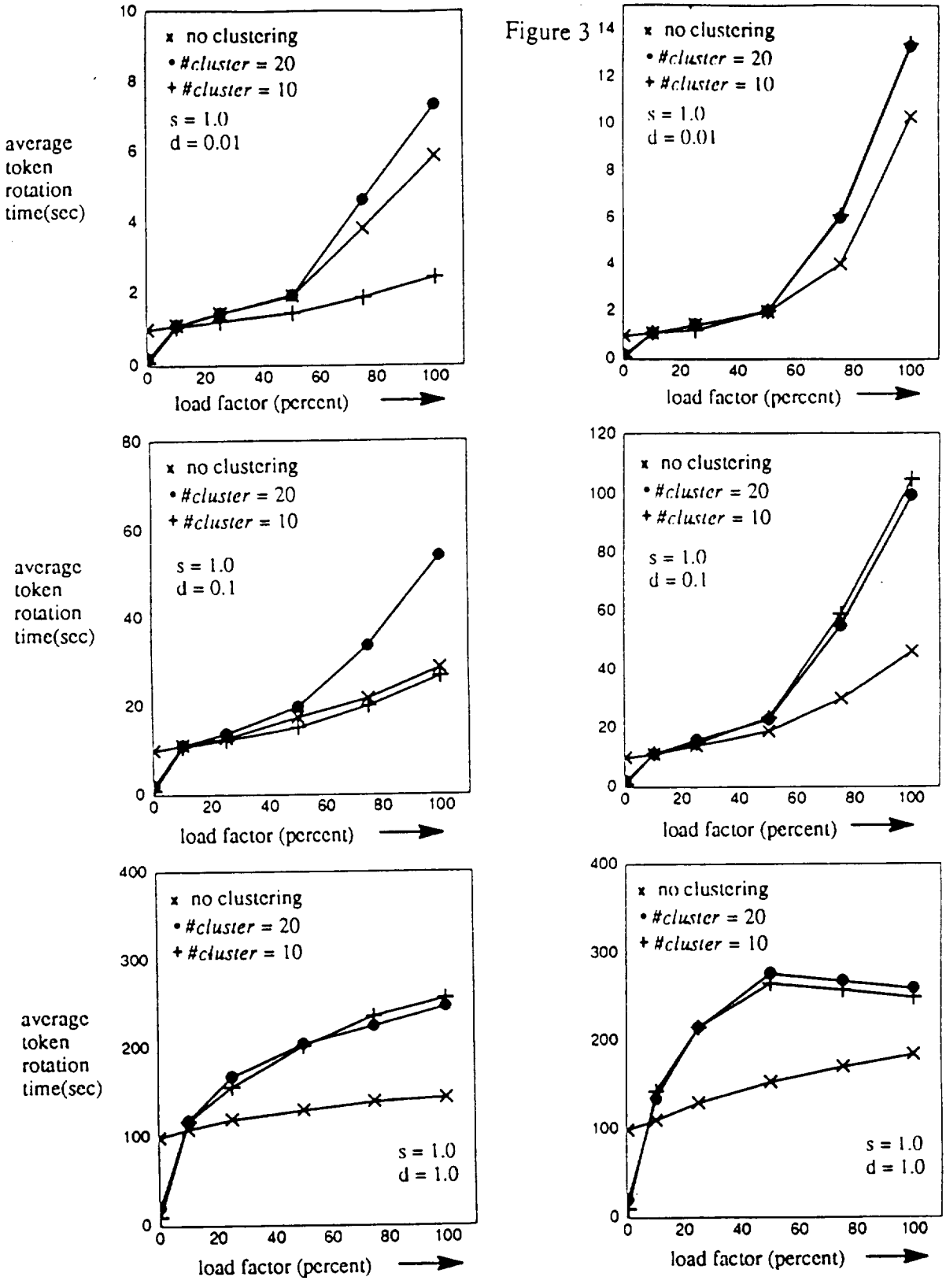
[6] S. Peterson and J. Kearns, "Performance evaluation of network mutual exclusion protocols," Computer Science Tech. Rep. 89-2, The College of William and Mary, 1989.

[7] M. Raynal, *Distributed algorithms and protocols*, John Wiley & sons, 1988.

[8] G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks," *Commun. ACM*, Vol. 24, No. 1, pp. 9-17, Jan. 1981.

[9] I. Suzuki and T. Kasami, "A distributed mutual exclusion algorithm," *ACM Trans. Computer Systems*, Vol. 3, No. 4, pp. 344-349, Nov. 1985.

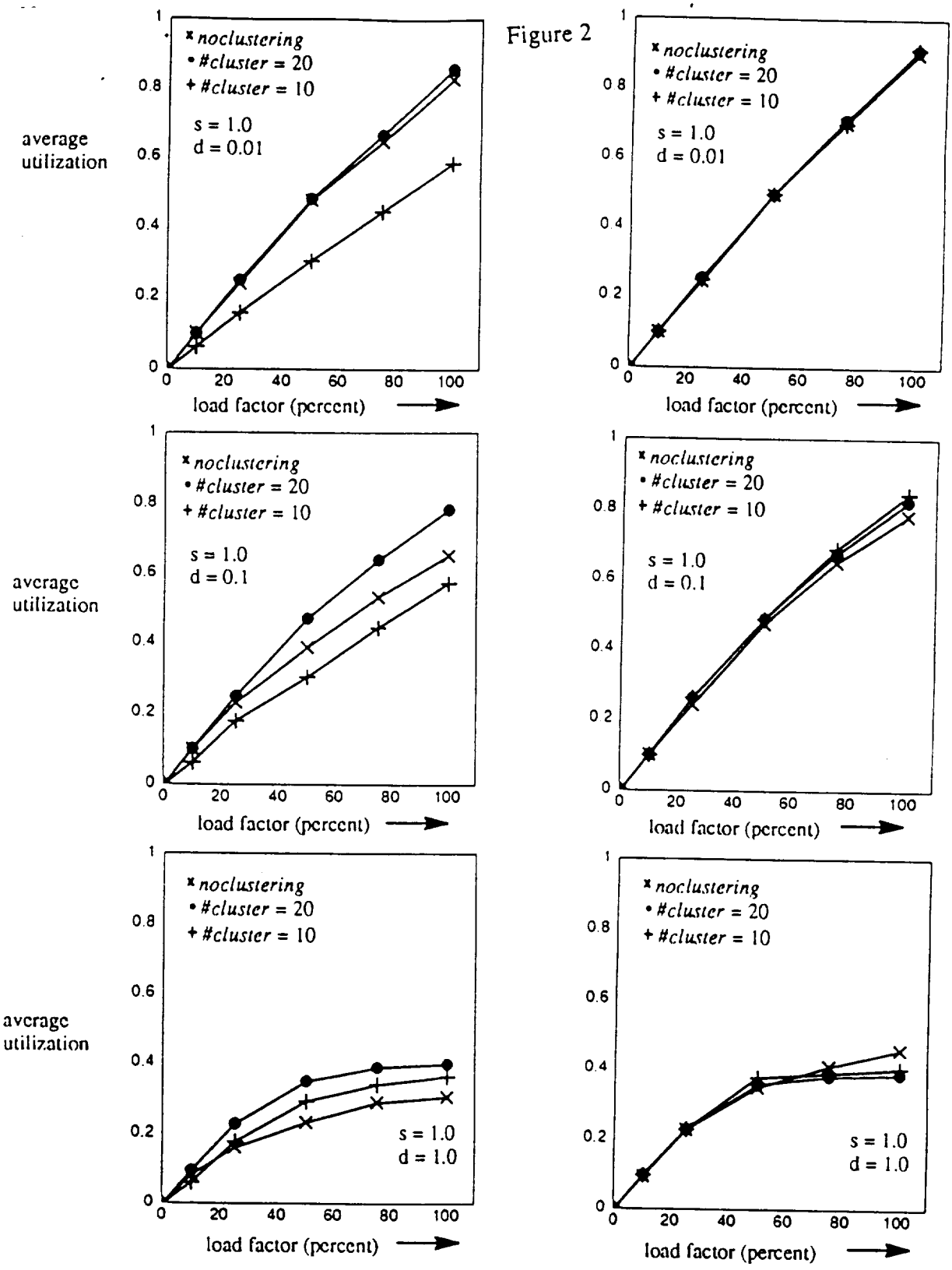
Figure 3



20% of the nodes take up 80% of the load;

Uniform load;

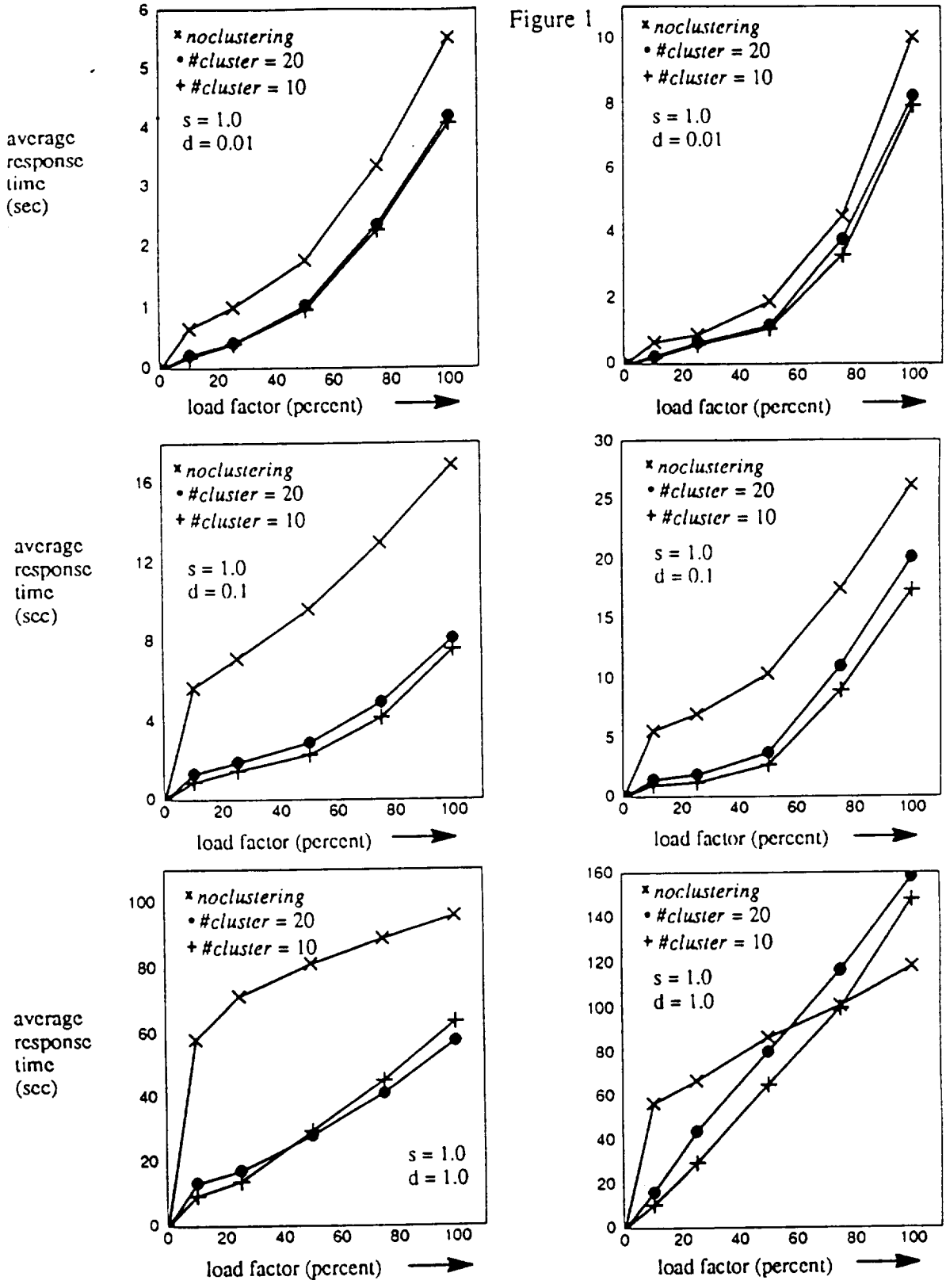
Figure 2



20% of the nodes take up 80% of the load;

Uniform load;

Figure 1



20% of the nodes take up 80% of the load;

Uniform load;

To Appear in IEEE Trans. on Knowledge and Data Engineering

Storage Efficient and Secure Replicated Distributed Databases

Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Abstract

Data availability and security are two important issues in a distributed database system. Existing schemes achieve high availability at the expense of higher storage cost, and data security at the expense of higher processing cost. In this paper, we develop an integrated methodology which combines the features of some existing schemes dealing with data fragmentation, data encoding, partial replication, and quorum consensus concepts to achieve storage efficient, highly-available, and secure distributed database systems.

Index Terms — Availability, data encryption, data security, distributed databases, integrated methodology, processing costs, quorum consensus, replication, resiliency, storage cost.

1 Introduction

Data availability and security are two important requirements in distributed database system design. Data availability refers to the resiliency of database operations (read or write) to node and link failures in a system. For example, if an object is read (write) accessible in spite of r node failures, then it has a read (write) resiliency level of r [2, 4]. Data security has several definitions in the literature [5, 9]. In the context of distributed database systems, it may be defined as the maximum number of nodes that may be accessed without compromising the contents of a data object. This is also referred to as data confidentiality [2, 12, 13]. For example, if at least r nodes need to be accessed before the information in a data object is compromised, then its confidentiality is r. Since storage, processing and communication are three limited resources in a distributed system, the objective of a designer is to achieve the required availability and security with the minimum use of these resources.

In general, high data availability is achieved through replication of data objects in a system. Several replication schemes have been suggested to achieve high availability through replication of entire objects [1, 6, 7, 8, 11, 14]. Here, availability achieved at the expense of considerable storage and communication costs. These schemes do not address the data confidentiality issue.

To reduce the storage overhead, Paîris [10] proposed a protocol based on the concept of *witnesses*. In this scheme, copies of data objects are maintained at a few nodes, and a number of other nodes (called witnesses) maintain only a version number for the object. Thus, if there are w witnesses and $(n-w)$ data copies, then the storage overhead is about $(n-w)$ times the

size of the object. This storage reduction is achieved without compromising the fault-tolerance aspects of the protocol. However, the protocol places a number of restrictions on the node recovery process [3, 10]. As before, this scheme does not address the confidentiality issue.

To overcome the complexities in node recovery and yet reduce the storage overhead, Agrawal and El Abbadi [3] proposed a fragmentation method in which an object is divided into n fragments and m copies of each fragment are then distributed among n nodes with each node having m distinct fragments. In this case, the total storage overhead due to replication is $(m-1)$ times the size of the object. With $m < n$, this scheme can reduce storage overhead considerably while offering high availability. Once again this scheme does not explicitly address the confidentiality issue.

A scheme to implement data security in terms of confidentiality using data encoding and decoding techniques was proposed by Rabin [12]. In addition to offering security, this scheme incurs low storage overhead. The reduction in storage overhead is achieved at the expense of increased communication and processing overhead during read and write operations on the object. To improve the data availability offered by this scheme, Agrawal and El Abbadi [2] have integrated this technique with Gifford's quorum consensus protocol [7] and Wuu and Bernstein's [15] log-based update propagation scheme. However, they do not employ the data fragmentation concept as a means of reducing data storage without incurring additional processing overhead.

While each of the above mentioned schemes seem to offer some advantages in terms of data availability, data security, low storage overhead, low processing cost, or low communication cost, currently there is no single

scheme that has attempted to unify these concepts.

We analyze the costs and benefits of the concepts underlying the above mentioned schemes, indicate the need for an integration of concepts, and propose an outline of a design methodology that integrates these concepts.

2 Cost-Benefit Analysis of Underlying Concepts

We now analyze the concepts underlying schemes for data allocation and management in distributed database systems discussed above. For simplicity, the discussion is limited to a single data object x containing L characters or bytes of information.

2.1 Quorum Consensus

This concept is the basis for many of today's highly available systems [7, 14]. Here, copies of the object x are stored at $n[x]$ nodes. Each data item x is associated with a read quorum, $q_r[x]$, and a write quorum, $q_w[x]$. The read and write quorums must satisfy the following requirements:

$$1 \leq q_r[x] \leq n[x] \tag{1}$$

$$\lceil (n[x] + 1)/2) \rceil \leq q_w[x] \leq n[x] \tag{2}$$

$$n[x] < q_r[x] + q_w[x] \leq 2 \cdot n[x] \tag{3}$$

Here, a read operation is executed by accessing $q_r[x]$ copies of x and determining the most up-to-date copy using a version number associated with each copy. A copy with the highest version number is then accessed for read. Similarly, a write operation is executed by accessing $q_w[x]$ copies, determining the highest version number, and updating all the accessed copies with the new value as well as declaring them as the latest through a new version number.

Clearly, since the entire data object is replicated at $n[x]$ nodes, the storage cost is $L \cdot n[x]$. Since each read operation requires $O(q_r[x])$ messages, the communication cost for a read is $O(q_r[x])$. Similarly, the communication cost for a write is $O(q_w[x])$. Even though there is some processing involved in comparing the version numbers, it is considered negligible compared to other database operations on the data object. Looking at the confidentiality, since each copy contains an entire image of x, accessing any of the $n[x]$ copies will compromise security. Even though a copy may not always be up-to-date, it still has the potential of compromising the data security. Thus, this scheme offers zero confidentiality. From the description of the read and write operations, it may be discerned that the read resiliency level is $n[x] - q_r[x]$ and the write resiliency level is $n[x] - q_w[x]$.

2.2 Data Fragmentation

In order to reduce the high storage cost associated with full copy replication, this technique suggests fragmenting the data object, and replicating the fragments among the nodes [3]. For example, if x is partitioned into $n[x]$ nonoverlapping fragments, and each of these fragments is replicated at $m[x]$ nodes ($m[x] < n[x]$), then the total storage required is $L \cdot m[x]$. However, since each single node may only contain $m[x]$ fragments, one may have to access more than one node to reconstruct a complete copy of x. While improving confidentiality is not the main objective of this concept, it does offer better confidentiality than the full copy replication. The exact measure of confidentiality depends on the distribution of fragments among the nodes. In order to reconstruct x, we need to access at most $n[x] - m[x] + 1$ nodes. This is also referred to as full copy equivalent of x, or $fce[x]$ [3]. Thus, assuming a read-one/write-all replica control, accessing any $fce[x]$ nodes

will enable read operations on x. Thus, the reduction in storage is achieved at the increased cost of communication for read operations. The read and write resiliencies offered by this concept depends on the replica control policy. For example, with a read-one/write-all policy, the read resiliency is $n[x] - fce[x]$ or $m[x] - 1$, and the write resiliency is zero. If we employ the quorum consensus concept, then the read resiliency is $n[x] - q_r[x]$ and the write resiliency is $n[x] - q_w[x]$ where $q_r[x]$ and $q_w[x]$ are constrained by the following:

$$fce[x] = n[x] - m[x] + 1 \tag{4}$$

$$fce[x] \leq q_r[x] \leq n[x] \tag{5}$$

$$\max\left(fce[x], \lceil (n[x]+1)/2 \rceil\right) \leq q_w[x] \leq n[x] \tag{6}$$

$$n[x] + fce[x] \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \tag{7}$$

Thus, with the quorum concensus concept, the fragmentation concept has achieved lower data storage at increased lower limit on $q_r[x]$. This implies lower read resiliency and higher communication cost for read operations.

2.3 Data Encoding and Decoding

Even though fragmentation technique has reduced the data storage requirement of a replicated system, it is still considerably higher than a single copy system. In addition, the confidentiality offered by this scheme is not directly controlled by the number of fragments. Instead it depends on the actual distribution of fragment copies among the nodes.

The data encoding and decoding concept is a scheme to directly control the confidentiality of a data object and achieve it at a substantially low storage overhead. One such scheme is suggested by Rabin [12]. Here, the

object x which is a sequence of L characters, $b_1, b_2, \ldots, b_L$, is segmented into sequences of length p. (If L is not a multiple of p, then x can be padded with null characters to make it a multiple of p). Denoting the ith segment of x as s_i, $x = s_1, s_2, \ldots, s_{(L/p)}$. These segments are now coded into n vectors $f_1, f_2, \ldots, f_n$ where $f_i = a_i \cdot s_1, a_i \cdot s_2, \ldots, a_i \cdot s_{(L/p)}$. Here, $\cdot$ denotes the vector dot product and a_i is a vector of length p. The n vectors, $a_1, a_2, \ldots a_n$ are chosen such that every subset of p different vectors are linearly independent. Rabin [12] describes a simple method to derive these vectors. The n coded vectors $f_1, f_2, \ldots, f_n$ are distributed to n nodes in the system. This completes the encoding procedure.

In order to reconstruct or decode x, we need to access p coded vectors from p nodes. Suppose vectors $f_1, f_2, \ldots, f_p$ are accessed for the reconstruction. If we denote $f_i = c_{i1}, c_{i2}, \ldots, c_{i(L/p)}$, and let A be a $p \times p$ matrix such that ith row of A is a_i, then the original segment s_i of x can be obtained by a matrix multiplication operation as follows: $s_i = A^{-1} \cdot [c_{1i}, c_{2i}, \ldots, c_{pi}]^{\mathrm{T}}$. Since $x = s_1, s_2, \ldots, s_{(L/p)}$, we have reconstructed the object x by decoding the p vectors.

Since each of the n coded vectors is L/p characters in size, the total storage required is $L \cdot n/p$. In addition, we need to store the a vectors requiring an additional n^2 storage. Since $L >> n$, we ignore this component. By a proper choice of p, the total storage can be considerably reduced. Looking at the confidentiality, since the information in x cannot be decoded without accessing p nodes, this scheme offers a confidentiality of p.

If we assume a read-one/write-all policy, then a read operation on x requires to access p nodes. In addition, it involves $2 \cdot L \cdot p$ arithmetic operations for reconstruction of x. For large data objects, this may be a significant cost.

A write operation should update the coded vectors at all the n nodes. In addition, depending on the type of changes, it may involve up to $2 \cdot L \cdot n$ arithmetic operations for encoding. Also, the read resiliency is $n - p$ and write resiliency is zero.

The write resiliency can be considerably improved by employing the quorum consensus concept [2]. Here, the read and write quorums must satisfy the following requirements:

$$p \leq q_r[x] \leq n[x] \tag{8}$$

$$\max(p, \lceil (n[x] + 1)/2 \rceil) \leq q_w[x] \leq n[x] \tag{9}$$

$$n[x] + p \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \tag{10}$$

With the quorum consensus, the read resiliency is increased to $n[x] - q_r[x]$, the write resiliency to $n[x] - q_w[x]$, and the confidentiality remains at p as before.

2.4 Log-based Information Propagation

In much of the earlier discussions, we assumed explicit read/write requests flowing between nodes using explicit message passing. Here, higher values of read and write quorums implies increased number of messages to execute read and write operations respectively. Wuu and Bernstein [15] proposed an efficient log-based propagation mechanism in which the only means of node-to-node communication is through the propagation of logs. Agrawal and El Abbadi adopted this scheme for replicated databases [2]. When a node intends to execute a read operation on x, it writes $r[x]$-event into its log. Each log, and hence the read-event, ultimately reaches other sites. When a site learns of $r[x]$-event and decides to be in the quorum, it writes $ok(r[x])$-event into its log along with the version number and the value of x.

When the original site learns of the required majority of $ok(r[x])$-events, by reading the logs from other sites, it determines the latest value of x. The read operation is now complete.

In the case of a write operation, the node intending to execute the operation requests for version numbers of x by writing $v[x]$-event in its log which is ultimately propagated to others. When a node reads this request and decides to participate in the quorum, it writes its version number and the value of x in its log along with *ok*-event. When the original node learns of the majority of *ok*-events, it updates the value and writes $w[x]$-event in its log.

As a consequence of employing this propagation technique along with the data fragmentation technique, (7) can be relaxed as follows [3]:

$$n[x] + 1 \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \tag{11}$$

Similarly, when this technique is combined with the data encoding scheme, (10) can be relaxed as follows [2]:

$$n[x] + 1 \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \tag{12}$$

Thus, the log-based propagation has improved the read and write resiliencies when integrated with the data fragmentation and data encoding schemes.

3 An Integrated Approach

While each of the above mentioned concepts have certain benefits, these are achieved at the expense of additional processing cost, storage cost, or restrictions on read and write availabilities. Some of these restrictions may be overcome through a proper integration of these techniques. Before we

present our integrated approach, we illustrate its need through the following examples. In each of these examples, given the system requirements, we have to suggest an appropriate data management scheme. Here, for simplicity, we express the resiliency in terms of read and write quorums (q_r, q_w).

Example 1: Requirements: Number of nodes = 10; Storage $\leq$ 5.L; confidentiality $\geq$ 4; resiliency: $q_r \leq 4$, $q_w \leq 8$.

Suppose we apply the fragmentation technique [3] along with the log-based propagation [15] and the quorum consensus [7], then x can be partitioned into ten fragments, each of which can be replicated at five or less sites (due to the storage constraint). Hence, $fce \geq 6$ (4) and $q_r \geq 6$ (5). Clearly, this is an unacceptable solution since it does not satisfy the read resiliency requirement.

Suppose we apply the data encoding scheme with log-based propagation [15] and quorum consensus [2], then the storage constraint restricts p to be $2 \leq p \leq 10$. Since we need a confidentiality of at least 4, p is further constrained as $4 \leq p \leq 10$. Accordingly, $4 \leq q_r \leq 10$ (8). To satisfy the read resiliency requirement, we choose $p = 4$ and $q_r = 4$. Hence, $7 \leq q_w \leq 10$ ((9) and (12)). Thus, using the data encoding method we are able to obtain a data management scheme which satisfies all the design requirements.

Example 2: Requirements: Number of nodes = 10; Storage $\leq 6 \cdot L$; confidentiality $\geq$ 3; resiliency: $q_r \leq 5$, $q_w \leq 7$; processing cost: arithmetic operations per read $\leq 4 \cdot L$.

Applying the fragmentation technique with log-based propagation and quorum consensus, we find $m \leq 6$ (storage constraint), and hence $fce \geq 5$.

Accordingly, $q_r \geq 5$ (5). To satisfy the read resiliency requirement, let $fce = q_r = 5$. Also, $6 \leq q_w \leq 10$ ((6) and (11)). Since $q_r = 5$, it is possible to distribute the fragment copies among nodes so that the confidentiality requirement of x is met. Hence, this solution is acceptable.

If we apply the data encoding scheme along with the log-based propagation and quorum consensus, the storage constraint restricts p as $2 \leq p \leq 10$. With a confidentaility of three, p should be at least 3. Hence, $3 \leq p \leq 10$. However, this implies that each read operation involves at least $6 \cdot L$ arithmetic operations. This is unacceptable.

Example 3: Requirements: Number of nodes = 10; Storage $\leq 4 \cdot L$; Availability: $q_r \leq 4$; processing cost: arithmetic operations per read $\leq 4 \cdot L$.

Applying the fragmentation technique with log-based propagation and quorum consensus, we find $m \leq 4$ (storage requirement), and hence $fce \geq 7$. Accordingly, $q_r \geq 7$. This is an unacceptable solution.

Applying the data encoding scheme with log-based propagation and quorum consensus, we find $p \geq 3$ (storage requirement). However, this implies that each read operation requires at least $6 \cdot L$ arithmetic operations. This is unacceptable.

As an alternate, suppose we first partition x into (say) ten fragments $(F_1, F_2, \ldots, F_{10})$ and apply the data encoding scheme on each fragment with $p = 2$ and $n = 4$. Thus, each fragment F_i results in 4 vectors $(f_{i,1}, f_{i,2}, f_{i,3}, f_{i,4})$, any two of which are sufficient to reconstruct it. In addition, if we make two copies of each vector, then we have 8 vector copies per fragment $(c_{i,1,1}, c_{i,1,2}, \ldots, c_{i,4,1}, c_{i,4,2})$. In total, we will have 80 coded vectors to be distributed among 10 nodes (or 8 per node). The storage cost

Nodes									
1	2	3	4	5	6	7	8	9	10
$c_{1,1,1}$	$c_{1,2,1}$	$c_{1,3,1}$	$c_{1,4,1}$		$c_{1,1,2}$	$c_{1,2,2}$	$c_{1,3,2}$	$c_{1,4,2}$	
	$c_{2,1,1}$	$c_{2,2,1}$	$c_{2,3,1}$	$c_{2,4,1}$		$c_{2,1,2}$	$c_{2,2,2}$	$c_{2,3,2}$	$c_{2,4,2}$
$c_{3,4,2}$		$c_{3,1,1}$	$c_{3,2,1}$	$c_{3,3,1}$	$c_{3,4,1}$		$c_{3,1,2}$	$c_{3,2,2}$	$c_{3,3,2}$
$c_{4,3,2}$	$c_{4,4,2}$		$c_{4,1,1}$	$c_{4,2,1}$	$c_{4,3,1}$	$c_{4,4,1}$		$c_{4,1,2}$	$c_{4,2,2}$
$c_{5,2,2}$	$c_{5,3,2}$	$c_{5,4,2}$		$c_{5,1,1}$	$c_{5,2,1}$	$c_{5,3,1}$	$c_{5,4,1}$		$c_{5,1,2}$
$c_{6,1,2}$	$c_{6,2,2}$	$c_{6,3,2}$	$c_{6,4,2}$		$c_{6,1,1}$	$c_{6,2,1}$	$c_{6,3,1}$	$c_{6,4,1}$	
	$c_{7,1,2}$	$c_{7,2,2}$	$c_{7,3,2}$	$c_{7,4,2}$		$c_{7,1,1}$	$c_{7,2,1}$	$c_{7,3,1}$	$c_{7,4,1}$
$c_{8,4,1}$		$c_{8,1,2}$	$c_{8,2,2}$	$c_{8,3,2}$	$c_{8,4,2}$		$c_{8,1,1}$	$c_{8,2,1}$	$c_{8,3,1}$
$c_{9,3,1}$	$c_{9,4,1}$		$c_{9,1,2}$	$c_{9,2,2}$	$c_{9,3,2}$	$c_{9,4,2}$		$c_{9,1,1}$	$c_{9,2,1}$
$c_{10,2,1}$	$c_{10,3,1}$	$c_{10,4,1}$		$c_{10,1,2}$	$c_{10,2,2}$	$c_{10,3,2}$	$c_{10,4,2}$		$c_{10,1,1}$

Table 1: Vector distribution for Example 3

is (4/2)*2*L or $4 \cdot L$. To read x, we need to reconstruct all ten fragments. Assuming that both the replicated copies of a vector are updated by a write, we should access 20 vectors to reconstruct x. This implies that we should access at least three nodes. By distributing the vectors in a cyclical fashion, as shown in Fig. 1, it can be seen that accessing any four nodes is sufficient to reconstruct x in a read operation. Thus, $q_r \geq 4$. We choose $q_r = 4$. Also, since $p = 2$ the arithmetic operations required for read operation on x are $4 \cdot L$.

This example illustrates the need for combining the data fragmentation and decoding techniques in the allocation of x among the nodes in a distributed database system.

4 Proposed methodology

The examples in the previous section indicate that data fragmentation and data encoding are not mutually exclusive alternates. In fact, these should be used as complimentary concepts. Here, we describe an outline of a method-

ology to integrate these concepts.

Problem: Given the storage, security, processing, and resiliency requirements on a data object x, determine a data allocation and processing scheme.

Step 1. Ignoring the processing cost constraints, determine a value of p (in the data encoding scheme) so as to satisfy the confidentiality and resiliency requirements in (8), (9), and (12). (If the log-based propagation is inappropriate, use (10) instead of (12).)

- If no such value of p can be found, then conclude that the concepts considered in this paper are inadequate to solve the problem.
- Otherwise, using p, determine the processing overhead associated with read and write operations in data encoding and decoding. If the processing requirements are met, then the desired solution is obtained by using data encoding, quorum consensus, and log-based propagation (if necessary) with the specified p and read/write quorums. Otherwise, go to Step 2.

Step 2. To determine the suitability of data fragmentation as a means to reduce processing cost, determine the value of m so as to satisfy the resiliency and confidentiality constraints while keeping the storage overhead within limits ((4)-(7)). The storage overhead, in this case, is given by $m \cdot L$ plus any storage needed for storing logs if log-based propagation is used. Since the processing overhead in the reconstruction and updation of x is minimal, it can be ignored.

- If no such value of m can be found, then conclude that data fragmentation with quorum consensus and log-based propagation

(if (11) is used instead of (7) for q_w) are insufficient to solve the problem and go to Step 3.

- Otherwise, the solution is obtained with replication specified by m and quorums specified by (4)-(7) (or (11) if log-based propagation is used). Since, in this case, confidentiality depends on the actual distribution of copies among the nodes, and since optimal allocation is an NP-hard problem, employ a heuristic algorithm to determine the allocation and the associated confidentiality. If no such distribution is found, go to Step 3.

Step 3. Here, we find both the storage and processing constraints to be significantly severe to obtain a solution using either data encoding or data fragmentation. In this step, we employ both of them simultaneoulsly. For simplicity, assume that x is partitioned to n fragments, where n is the number of nodes in the system. We need to choose values for

- p, n': the encoding parameters. Each fragment is encoded into n' vectors ($n' \leq n$) so that any p are sufficient to reconstruct it. The read and write quorums for a fragment are dictated by (8)-(10). (Use (12) if log-based propagation is used.)
- m: number of copies of each coded vector of a fragment.
- Distribution of vector copies among the nodes in the system.

Here, the storage requirement of the coded vectors is given by $m \cdot n'/p \cdot L$. The processing requirement for reconstruction (or read) is given by $2 \cdot p \cdot L$. The offered resiliency and confidentiality now depend on the distribution of vector copies. If no single node has access to more than one vector of a given fragment, then the confidentiality of a

single fragment is p. However, the confidentiality of x could be much higher. Similarly, even though the minimum values of q_r and q_w are dictated by (8)-(10), these could be higher for x, and depend on the distribution. In general,

Step 3a. Determine values of p, n', and m so as to satisfy the storage and processing requirements.

Step 3b. Determine a distribution scheme satisfying the resiliency and confidentiality requirements. Since data distribution, in general, is a NP-hard problem, employ heuristic schemes such as symmetric or cyclic distributions.

Step 3c. If Step 3b cannot find a solution, choose alternate values for the parameters and repeat Step 3b.

Due to the heuristic nature of the distribution algorithm, a solution may not be found even if it actually exists. Handling these problems is beyond the scope of this paper.

5 Conclusion

The main objective of this paper is to illustrate the need for integration of tools in database design. In particular, this paper dealt with the problem of data allocation in distributed systems using data fragmentation, data encoding, partial replication, and quorum consensus as basic techniques. While several papers in literature have described the efficacy of each of these techniques, in this paper we made an attempt to integrate them. First, we have illustrated the need for integration, and then outlined a procedure for integration. Future work should concentrate on optimizing the procedure.

ACKNOWLEDGEMENT

The author would like to thank the anonymous referees for their careful reading of the manuscript and for their comments. This work is supported in part by a grant from NASA Langley Research Center, NAG-1-1114.

References

[1] A. El Abbadi and S. Toueg, "Maintaining availability in partitioned replicated databases," *ACM Trans. Database Systems*, vol. 14, no. 2, pp. 264-290, June 1989.

[2] D. Agrawal and A. El Abbadi, "Integrating security with fault-tolerant distributed databases," *Computer Journal*, vol. 33, no. 1, pp. 71-78, February 1990.

[3] D. Agrawal and A. El Abbadi, "Storage efficient replicated databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, no. 3, pp. 342-352, September 1990.

[4] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing vote and quorum assignments for reading and writing replicated data," *IEEE Trans. Knowledge and Data Engineering*, vol. 1, no. 3, pp. 387-397, September 1989,

[5] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.

[6] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841-860, Oct. 1985.

[7] D. K. Gifford, "Weighted voting for replicated data," in *Proc. Seventh Symp. Oper. Syst. Principles,* Dec. 1979, pp. 150-159.

[8] S. Jajodia and D. Mutchler, "Dynamic voting algorithms for maintaining the consistency of a replicated database," *ACM Trans. on Database Systems*, vol. 15, no. 2, pp. 230-280, June 1990.

[9] T. F. Lunt, D.E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, "The SeaView security model," *IEEE Trans. Software Engineering*, vol. 16, pp. 593-607, June 1990.

[10] J.-F. Paŕis, "Voting with witnesses: A consistency scheme for replicated files," in *Proc. 6th Conf. Dist. Comp. Syst.*, June 1986, pp. 606-612.

[11] J.-F. Paŕis and D. E. Long, "Efficient dynamic voting algorithms," in *Proc. Fourth IEEE Intl. Conf. on Data Engineering*, Feb. 1988, pp. 268-275.

[12] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335-348, April 1989.

[13] B. Randell and J. Dobson, "Reliability and security issues in distributed computing systems," in *Proc. Fifth Symp. on Reliability in Distributed Software and Database Systems,* 1986, pp. 113-118.

[14] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180-209, June 1979.

[15] G. T. J. Wuu and A. J. Bernstein, "Efficient solutions to the replicated log and dictionary problems," in Proc. 3rd Symp. Principles of Distributed Computing, 1984, pp. 233-242.

Order Statistics Based Modeling of Gracefully Degrading Computing Systems

N. R. Chaganty and R. Mukkamala
Old Dominion University, Norfolk, Virginia 23529-0162, USA.

Key Words— Fault-tolerance, Failure Rate, Graceful degradation, Hardware redundancy, Mean computations before failure, Mean time to failure, Order statistics, Reliability.

Abstract

Much of the existing work on gracefully degrading computing systems are only applicable when the life distribution for the system components is exponential. In this paper, we develop an order-statistics based model for these systems. We show the generality of this model by deriving expressions for system reliability, mean-time-to-failure, and mean-computations-before-failure. In addition, we derive some interesting properties for the failure rate of such a system . In particular we show that the failure rate of a gracefully degrading system with i.i.d. DFR components is also DFR if the coverage probability is less than 1/2. This generalizes a well known result for series systems.

1 INTRODUCTION

With the advances in VLSI technology, it is now possible to build systems with replicated components at almost no additional cost. For example, it is

possible to obtain a computing system with replicated processors (or multiprocessors), replicated memory modules, or redundant interconnections [8]. Depending on the application, component redundancies may be used to improve the response time of the system (through parallelization), to improve the fault-tolerance (or reliability) of the system, or both. Among these, the third type, generally referred to as gracefully degrading systems, are more attractive since they combine the efficiencies due to parallelism with the fault-tolerance properties of standby-redundant systems [3, 9]. In addition to increased hardware reliability, redundant hardware components may be also used to enhance software reliability with schemes such as triple-modular redundancy and N-version programming [1, 5].

Even though the reliability of individual components in a system is considerably high, due to technological advances in design and manufacturing, the extremely high number of components in a system may result in significantly low system reliability. In addition, with the increase in need for ultra-reliable systems, such as in space applications, the study of system reliability has become important. In this context, we look at the performance of gracefully degrading systems.

Gracefully degrading systems are a class that include the series and parallel systems at the extremes. These systems may use all redundant components to execute tasks, i.e., all failure-free components are active. When a component failure is detected, these systems attempt to reconfigure to a

system with one fewer components. These systems can be represented as falling somewhere between the extremes of ultra-reliable systems and high performance parallel systems in terms of the trade-off between performance and reliability gained by the use of redundancy [3, 9].

Much of the existing studies on gracefully degrading systems concentrate on the evaluation of performance metrics such as response time (e.g., time of task completion) and reliability (e.g., probability of task completion). For example, Borgerson and Freitas introduced a model for analyzing the reliability of these systems [4]; Beaudry has analyzed and studied the performance characteristics of these system using Markov model and demonstrated that the gracefully degrading systems have a higher probability of executing a long computing mission than the corresponding standby redundant systems [3]; further evaluation of gracefully degrading systems from the viewpoint of generalized reliability measures has been done by Osaki [9]; and Koren et al derive expressions for the performance of a gracefully degrading multiprocessor system with multistage interconnection system [8]. Much of this work concentrates on investigating the average behavior of the system such as the average reliability or the average time for task completion assuming exponential life distributions for system components. For example, Beaudry's method of Markov chains is valid only under exponential life distributions, and is inapplicable for general distributions. These observations are also valid with other previous work mentioned above. In this paper we present

a fairly general model for gracefully degrading systems through the use of order statistics [7] and show how one can derive various reliability measures for systems consisting of components with arbitrary life distributions.

There is considerable amount of literature in the theory of reliability that is devoted to studying the behavior of the failure rate of systems [2, 11]. Specifically, several studies dealt with relating the failure rate behavior of a system to the failure rate behavior of its components. For example, a classic result states that a series system of n independent and identically distributed (i.i.d.) components has decreasing failure rate (DFR) if the components possess DFR property. Similarly, a parallel system of n i.i.d. components is shown to have an increasing failure rate (IFR) if the components possess IFR property [10, 11]. The concepts of failure rate and IFR, DFR are defined precisely in Section 4. In this paper, we derive some important properties for the failure rate of the gracefully degrading systems.

This paper is organized as follows. In section 2 we describe the order-statistics based model of the gracefully degrading systems. Derivations for the three chosen metrics is described in Section 3. Section 4 contains the results on failure rate properties. Finally, Section 5 has some concluding remarks.

2 Order-statistics based Model of a Gracefully Degrading System

Gracefully degrading systems are a class that include the series and parallel systems at the extremes. These systems use all components to execute tasks, i.e., all failure-free components are active. The model for the gracefully degrading computing system can be described as follows [9]:

1. The system has n i.i.d. components.

2. The life length of each unit has an arbitrary Cdf, $F(t)$ and Pdf $f(t)$.

3. A system is in state i when there are i failed components, $0 \leq i \leq (n-1)$.

4. c_i is the coverage probability when a system is in state $i-1$, $1 \leq i \leq (n-1)$.

5. $T_{(1)}, T_{(2)}, \ldots, T_{(n)}$ are the n-order statistics of the life lengths.

6. T is the life time of the system.

Initially the system starts out with all n components working and the state of the system is taken to be 0. When the component failure is detected at time $T_{(1)}$, the system reconfigures itself to a system with $(n-1)$ components with probability c_1 and fails with probability $1-c_1$. The state of the system is now 1. In general when the system is in state $(i-1)$ and the ith failure

occurs at time $T_{(i)}$, the system reconfigures itself and starts working in state i with probability c_i and fails with probability $(1-c_i)$. Under these conditions, the life time distribution of a gracefully degrading system may be expressed in terms of the component life times and coverage probabilities as

$$T = \begin{cases} T_{(1)} & \text{with probability } 1 - c_1 \\ T_{(2)} & \text{with probability } c_1(1 - c_2) \\ \vdots & \vdots \\ T_{(n-1)} & \text{with probability } c_1 c_2 \ldots c_{n-2}(1 - c_{n-1}) \\ T_{(n)} & \text{with probability } c_1 c_2 \ldots c_{n-1} \end{cases} \quad (1)$$

The coverage probability c_i indicates the probability with which the system can reconfigure when the ith failure has occurred. For example, in the case where $c_i = 0$, $1 \leq i \leq (n-1)$, the gracefully degrading system corresponds to a series system and $c_i = 1$ for all $1 \leq i \leq (n-1)$ corresponds to a parallel system of n i.i.d. components.

3 Evaluation of Performance Metrics

In this section, we use the order statistics model to derive expressions for three important performance metrics in fault-tolerant computing: system reliability, the mean-time-to-failure, and the mean computation before failure.

The reliability of a system, $\overline{R}(t)$, is defined as the probability with which the system is operable at least until time t. In other words, $\overline{R}(t)$=Pr[$t_F >$ t|initial system state], where t_F is the time of system failure. Using this definition, and the order statistics model described above, the reliability of

a gracefully degrading system can be computed as

$$\begin{aligned}\overline{R}(t) &= (1-c_1)P(T_{(1)}>t) + c_1(1-c_2)P(T_{(2)}>t) + \ldots + c_1c_2\ldots c_{n-1}P(T_{(n)}>t) \\ &= P(T_{(1)}>t) + c_1P(T_{(1)}<t<T_{(2)}) + \ldots + c_1c_2\ldots c_{n-1}P(T_{(n-1)}<t<T_{(n)})\end{aligned} \tag{2}$$

Let $\overline{F}(t) = 1 - F(t)$ be the survival function of a single component, then we can easily see that

$$P(T_{(k)} < t < T_{(k+1)}) = \binom{n}{k} [F(t)]^k \left[\overline{F}(t)\right]^{n-k} \tag{3}$$

Hence,

$$\overline{R}(t) = \sum_{k=0}^{n-1} \left(\prod_{i=0}^{k} c_i\right) \binom{n}{k} [F(t)]^k \left[\overline{F}(t)\right]^{n-k} \tag{4}$$

where $c_0 = 1$.

In the case where $c_1 = c_2 = \ldots = c_n = c$, we get

$$\begin{aligned}\overline{R}(t) &= \sum_{k=0}^{n-1} \binom{n}{k} [cF(t)]^k \left[\overline{F}(t)\right]^{n-k} \\ &= \left[cF(t) + \overline{F}(t)\right]^n - [cF(t)]^n\end{aligned} \tag{5}$$

which agrees with Osaki's result [9].

Let us now consider a second important performance metric for a gracefully degrading computing system: the mean time to failure or MTTF. By definition it represents expected life of a system before it fails. From the definition of the life time of the system (T) in the order statistics model and the

above definition, MTTF for a gracefully degrading computing system may be expressed as:

$$\begin{aligned} MTTF &= (1-c_1)E(T_{(1)}) + c_1(1-c_2)E(T_{(2)}) + \ldots + c_1c_2\ldots c_{n-1}E(T_{(n)}) \\ &= E(D_1) + c_1E(D_2) + c_1c_2E(D_2) + \ldots + c_1c_2\ldots c_{n-1}E(D_n) \quad (6) \end{aligned}$$

where $T_{(0)} = 0$ and $D_k = T_{(k)} - T_{(k-1)}$ is known as the spacing between the $(k-1)$th failure and the kth failure. If $c_1 = c_2 = \ldots = c_{n-1} = c$, then

$$MTTF = \sum_{k=0}^{n-1} c^k E(D_{k+1}) \quad (7)$$

It should be noted here that several results on order statistics currently exist in literature [7]. One could use these results to compute $\overline{R}(t)$ and $MTTF$ for arbitrary component life distributions.

If the life distribution of each component is exponential with parameter λ, then $E(D_k) = \frac{1}{\lambda(n-k+1)}$. Hence,

$$MTTF = \frac{1}{\lambda}\sum_{k=0}^{n-1} \frac{c^k}{(n-k)} \quad (8)$$

This agrees with Osaki's result [9].

We now consider an important performance metric relevant to computing systems which are likely to execute jobs that can store (and later retrieve) the intermediate results obtained by a task prior to the system failure. Database applications which store the state of a database system at intermediate points, known as checkpoints, are an example of such jobs [6]. This procedure is also common among tasks that have long computation times.

In such cases, when a system restarts after a failure, it restores the system (or task) state to the one indicated by the last checkpoint, and continues execution from that point. In this context, computing the expected progress between failures or mean computations before failure (MCBF) will enable us to compute the total time required to complete a given task. We now derive an expression for MCBF using the order statistics model.

Since we are dealing with a gracefully degrading computing system, it is important to recognize the fact that different states of the system have different computational capabilities. For example, a state with n components is faster than a state with $n-1$ components, and so on. If state i corresponds to i failed components and $n-i$ active components, then we let α_{n-i} be the computation capacity of the system in state i, $0 \leq i \leq (n-1)$. If the system starts in state 0, then using (1), we get the following expression for mean computation before failure (MCBF):

$$\begin{aligned} MCBF &= (1-c_1)\alpha_n E(T_{(1)}) + c_1(1-c_2)\left(\alpha_n T_{(1)} + \alpha_{n-1}E(T_{(2)} - T_{(1)})\right) + \ldots \\ &\quad + c_1 c_2 \ldots c_{n-1}\left(\alpha_n E(T_{(1)}) + \alpha_{n-1}E(T_{(2)} - T_{(1)}) + \ldots + \alpha_1 E(T_{(n)} - T_{(n-1)})\right) \\ &= \alpha_n E(D_1) + c_1\alpha_{n-1}E(D_2) + c_1c_2\alpha_{n-2}E(D_2) + \ldots + c_1c_2\ldots c_{n-1}\alpha_1 E(D_n) \\ &= \sum_{k=0}^{n-1}\left(\prod_{i=0}^{k-1} c_i\right)\alpha_{n-k}E(D_{k+1}) \end{aligned} \tag{9}$$

where $D_k = T_{(k)} - T_{(k-1)}$ and $c_0 = 1$.

Again, if the life distribution of the components is exponential with parameter λ, then D_k has an exponential distribution with parameter $(n-k+$

1)λ. Hence,

$$MCBF = \frac{1}{\lambda}\sum_{k=0}^{n-1}\left(\prod_{i=0}^{k-1} c_i\right)\frac{\alpha_{n-k}}{n-k} \tag{10}$$

If $\alpha_i = \alpha \cdot i$ and $c_1 = c_2 = \ldots = c$, then

$$\begin{aligned} MCBF &= \frac{\alpha}{\lambda}\sum_{k=0}^{n-1} c^k \\ &= \frac{\alpha(1-c^n)}{\lambda(1-c)} \end{aligned} \tag{11}$$

All the expressions for reliability measures that we derived for the exponential life distributions match with Beaudry's results [3].

4 Some Properties of the Failure Rate

As mentioned in the introduction, the failure rate of a system is an important fault-tolerant property of a system. For this reason, several theories expressing the failure rate of a system in terms of the component failure rates currently exist in literature. However, this metric has not yet attracted the attention of researchers dealing with gracefully degrading computing systems. We attempt to elicit such interest through this paper.

Since the life distribution function of a component has an arbitrary Cdf $F(t)$ and Pdf $f(t)$, then the failure rate of the component at time t is defined as $r(t) = f(t)/\overline{F}(t)$ [7]. The function $r(t)$ represents the probability intensity with which a component that is t-time units old will fail. Similarly the failure rate of the system at time t is defined as $r^*(t) = g(t)/\overline{R}(t)$, where $g(t)$ is the

Pdf of the life of the system and $\overline{R}(t)$ is the reliability of the system. We say the component (system) has IFR (increasing failure rate) property if $r(t)$ ($r^*(t)$) is increasing with t and DFR (decreasing failure rate) property if $r(t)$ ($r^*(t)$) is decreasing in t.

In general, for arbitrary c_i's there may not be any monotonic property (IFR or DFR) for the failure rate of a gracefully degrading system. However, in this section, we state and prove some monotonic properties of the failure rate of gracefully degrading systems in the special case where $c_i = c$ for all $1 \leq i \leq (n-1)$.

Using the above definitions of $r(t)$ and $r^*(t)$ and the order statistics based model described in Section 2, we can express the system failure rate of a gracefully degrading system as:

$$r^*(t) = nr(t)h(p_t) \tag{12}$$

where

$$p_t = \frac{cF(t)}{\left[cF(t) + \overline{F}(t)\right]} \tag{13}$$

$$h(p) = \frac{(1-p)\left(1-c+cp^{n-1}\right)}{1-p^n}. \tag{14}$$

It is interesting to note that we are able to express the failure rate of the system in terms of the failure rate of a component. The function $h(p_t)$ plays

a very important role in studying the failure rate of the system. Note that p_t is an increasing function of $t \geq 0$ and takes values between 0 and 1. Thus we only need to study the behavior of the function $h(p)$ for values of p between 0 and 1. Because of its practical importance, we state below as Proposition 1, the failure rate characteristics of a two component gracefully degrading system.

Proposition 1. For a gracefully degrading system of 2 components the following properties hold:

(a) If $c < \frac{1}{2}$, then $r(t)$ is DFR implies $r^*(t)$ is DFR.

(b) If $c = \frac{1}{2}$, then $r^*(t)$ is identically equal to $r(t)$.

(c) If $c > \frac{1}{2}$, then $r(t)$ is IFR implies $r^*(t)$ is IFR.

Proof: Substituting $n = 2$ and taking derivative of $h(p)$ in (14), we get

$$h'(p) = \frac{2c-1}{(1+p)^2}. \tag{15}$$

Therefore we can conclude that $h(p)$ is decreasing for $c < 1/2$, $h(p)$ is constant for $c = 1/2$, and $h(p)$ is increasing when $c > 1/2$. Since p_t is an increasing function of t, the result now follows from (12). •

Proposition 2. For a gracefully degrading system of n components the following properties hold:

(a) If $c \leq \frac{1}{2}$, then $r(t)$ is DFR implies $r^*(t)$ is DFR.

(b) If $c > \frac{1}{2}$, then $r(t)$ is DFR does not imply that $r^*(t)$ is DFR.

Proof: From (12), we can see that it suffices to show that $h(p)$ is decreasing for $c \leq \frac{1}{2}$ and for all $n \geq 1$. Taking derivative with respect to p in (14) and using the fact that $c \leq 1/2$ we get

$$h'(p) < \frac{-Q(n)}{2(1-p^n)^2}$$

where

$$Q(n) = \left(1 - p^{2n-2}\right) - (n-1)p^{n-2}\left(1-p^2\right).$$

We can easily verify that $Q(n) > 0$ for $0 \leq p \leq 1$ using induction argument. Therefore $h'(p) < 0$ and this proves (a). When $c > \frac{1}{2}$, and $n = 2$ as shown in Proposition 1, the function $h(p)$ is increasing in p. Thus we cannot say that $r^*(t)$ is DFR even if $r(t)$ is DFR. •

From Proposition 2 we can conclude that a series system ($c = 0$) of n i.i.d. DFR components is also DFR. The failure rate characteristics of a parallel system ($c = 1$) are stated in the next proposition. Although this is a well known result [11], we have included this proposition for the sake of completeness. Also, our proof is quite different from other known proofs.

Proposition 3. Consider a gracefully degrading system of n components. If $c = 1$ then the following properties hold:

(a) $r(t)$ is IFR implies $r^*(t)$ is IFR.

(b) $r(t)$ is DFR does not imply that $r^*(t)$ is DFR.

Proof: In the case $c = 1$ the derivative of the function $h(p)$ is given by

$$h'(p) = \frac{p^{n-2}\left[n(1-p) - (1-p^n)\right]}{(1-p^n)^2}. \tag{16}$$

Using induction argument we can show that $h'(p) > 0$ for all $n \geq 1$. This proves (a). Note that even if $r(t)$ is constant (hence DFR), $r^*(t)$ will be increasing in t since $h(p_t)$ is increasing in t. This shows (b). •

4.1 The Exponential Case

Since the exponential distribution is widely used to model the life distributions of electronic components and computer modules, it is only appropriate to discuss the behavior of the failure rate of a gracefully degrading system with exponential components in further detail. Let us assume that the system is composed of components which have exponentially distributed failure times with mean failure time equal to $1/\lambda$. Since $r(t) = \lambda$, using (12) we can rewrite the failure rate of the system as

$$r^*(t) = n\lambda h(p_t) \tag{17}$$

where p_t and $h(p_t)$ are as defined by (13) and (14). It is clear from (17) that the behavior of $r^*(t)$ is completely determined by the function $h(p_t)$. The

graph of $h(p_t)$ as a function of $F(t)$ for different values of c is shown in Figure 1. Note that $h(p_t) = 1 - c$ at $t = 0$ and converges to $1/n$ as $t \to \infty$. Also, $h(p_t)$ is decreasing in t for small values of c and increasing t for large values of c, as demonstrated in Proposition 2. The graph of $h(p_t)$ as a function of the coverage probability c for different values of $F(t)$ is shown in Figure 2. It clearly shows that at any point of time the failure rate is high for low coverage probability and vice versa, as we expect. Also the failure rate is a monotone decreasing function of the coverage probability.

5 Conclusion

In this paper, we have developed an order statistics based model to represent the fault-tolerant characteristics of a gracefully degrading computing system. This model has the advantage of being quite general and is usable under any arbitrary life time distribution assumptions for the components of a system. In addition, employing this model facilitates the use of several existing results in the area of order statistics. Using the proposed model, we have derived expressions for three performance metrics for gracefully degrading computing systems. These are verified with other results in literature for a special case of exponential life distributions for components. In addition, we have also stated and derived some properties related to the failure rate of the gracefully degrading systems. In particular, we have shown that the failure rate of a gracefully degrading system with i.i.d. DFR components is also DFR if the

coverage probability is less than 1/2. The three propositions stated in this paper generalize a number of already known results for series and parallel systems.

6 Acknowledgment

The first author's research was partially supported by the U.S. Army Research Office Grant number DAAL03-90-G-0103. The second author's research was partially supported by NASA Langley Research Center under grant NAG-1-1114. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] T. Anderson and P.A. Lee, *Fault Tolerance, Principles and Practice,* Prentice-Hall (1981).

[2] R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing,* Holt, Rinehart and Winston (1975).

[3] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Computers*, Vol C-27, p. 540-547 (1978).

[4] B. R. Borgerson and R. F. Freitas, "A Reliability Model for Gracefully Degrading and Standby-Sparing Systems," *IEEE Trans. Computers*, Vol. C-24, p. 517-525 (1975).

[5] L. Chen and A. Avizenis, "N-version programming: A fault-tolerance approach to reliability of software operation," *Proc. of FTCS-8*, p. 67-80 (1978).

[6] C. J. Date, *An Introduction to Database Systems, Vol I,* Addison Wesley (1986).

[7] H. A. David, *Order statistics,* John Wiley (1981).

[8] I. Koren and Z. Koren, "On gracefully degrading multiprocessors with multistage interconnection networks," *IEEE Trans. Reliability*, Vol. 38, p. 82-89 (1989).

[9] S. Osaki, "Performance/reliability measures for fault-tolerant computing systems," *IEEE Trans. Reliability*, Vol. R-33, p. 268-271 (1984).

[10] F. J. Samaneigo, "On closure of the IFR class under formation of coherent systems," *IEEE Trans. Reliability*, Vol. R-34, p. 69-72 (1985).

[11] K. S. Trivedi, *Probability Statistics with Reliability, Queuing, and Computer Science Applications,* Prentice-Hall (1982).

Figure 1. $h(p_t)$ as a function of $F(t)$

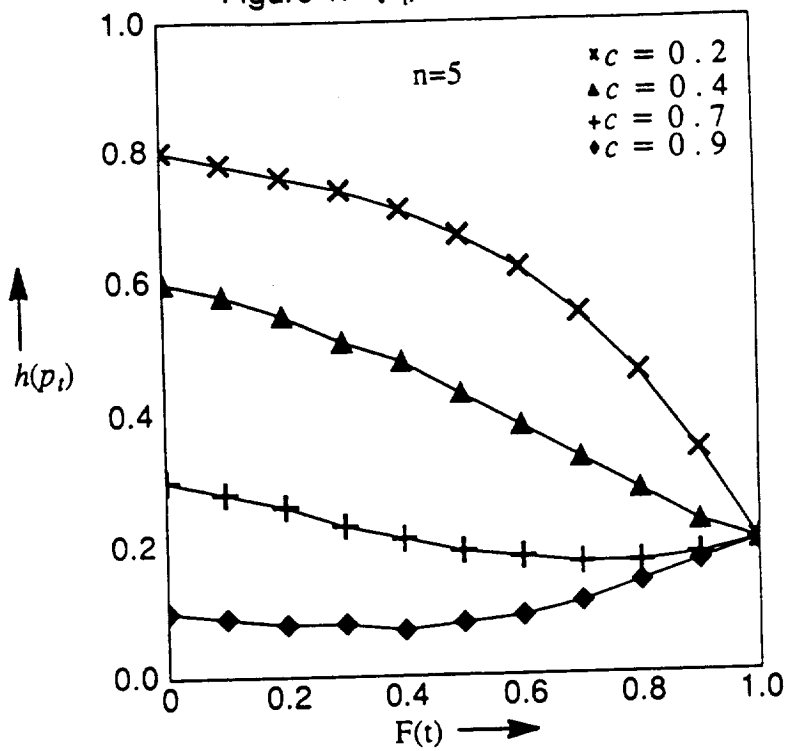
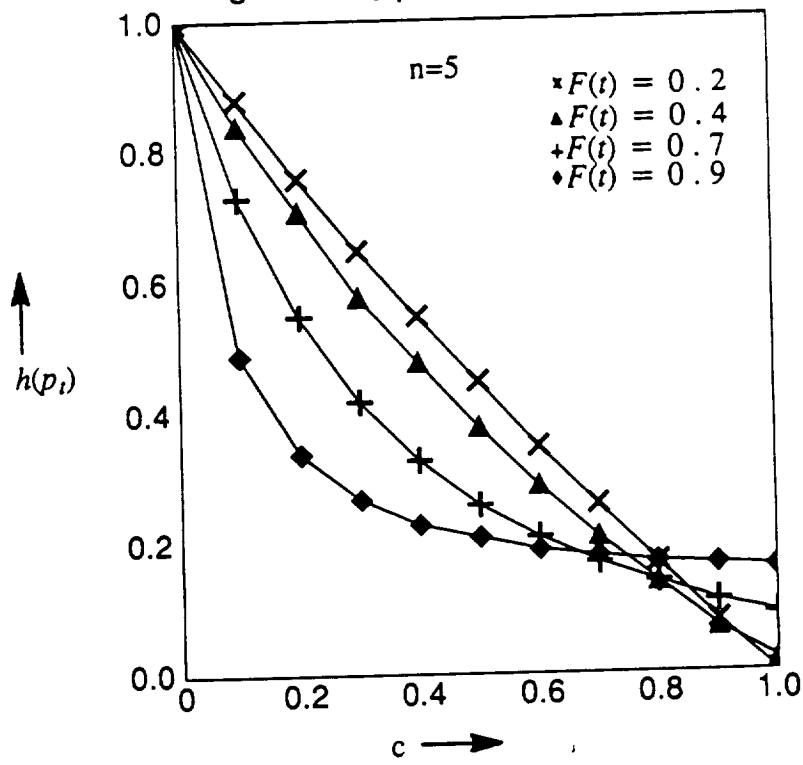


Figure 2. $h(p_t)$ as a function of c



DESIGN AND ANALYSIS OF DISTRIBUTED REAL-TIME ALGORITHMS

Yinghong Kuang
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

June 29, 1992

1 Introduction

A distributed system is a system with a number of processing elements and storage devices, connected together by a communication network [30]. Potentially, this makes a distributed system more powerful than a conventional, centralized one in two ways. First, it can be more reliable, because every function can be replicated several times. Second, a distributed system can do more work in the same amount of time, because many computations can be carried out in parallel. These two properties, fault tolerance and parallelism, give a distributed system the potential to be much more powerful than a centralized one.

The good features of the distributed systems also raise some problems. In distributed systems, generally there is no centralized unit in charge of the consistency control, since with this centralized unit we will lose the fault tolerance feature. The consistency control of distributed systems includes issues such as mutual exclusion, concurrency control, and replica control are well studied in literature [35, 40, 20, 29, 14, 15, 19, 25, 33, 3, 16]. The efficiency of these algorithms is generally measured in terms of the offered availability, the number of messages per request, the response time, and the throughput.

Real-time applications are those applications with strict requirements on the timing behavior of the system. The system that support the execution of real-time applications ensuring that the timing requirements are met are often referred to as real-time systems. Task scheduling is a vital component of real-time systems. Accordingly, this is a widely researched area resulting in real-time schedulers [49, 44, 28].

Due to the potential for parallelism and fault-tolerance in distributed systems, they are increasingly the preferred architectures for real-time sys-

tems. Here, by real-time distributed systems we mean those systems that consist of several dispersed concurrent processes that communicate with one another by exchanging messages and may have complex and time-critical interactions [5]. The complex interactions include sharing tasks among the nodes of a network, scheduling of tasks and maintaining consistency among processes [48]. The tasks as well as messages have timing constraints. In real-time uniprocessor systems the important activities are the deadlines of tasks, while in real-time distributed systems the important activities include both the deadlines of tasks and the deadlines of messages.

The existing approaches to distributed real-time systems include enhancements to operating systems [23, 28, 47], communication systems [12, 1] as well as distribution of subtasks according to resource allocation and fault tolerance requirement [24, 10, 49, 32, 47, 24]. While some studies start from the centralized implementation of real-time systems and attempt to introduce the distributed processing aspects in to the systems, others are starting from the general distributed systems and attempt to consider real-time systems as a viable application. Here, we chose the latter approach.

We propose to study the consistency control aspects of distributed real-time systems. In particular, we propose to concentrate on the three issues mentioned above: mutual exclusion, concurrency control and replica control. Since the existing solutions to these issues are mainly aimed at non-real time systems with emphasis on throughput and availability rather than on time-bounded responses, the proposed work will be useful in building distributed real-time systems. To achieve this task, we will attempt to establish a relationship between time boundedness, availability and concurrency in these systems. This may also require formalization of the semantics of concurrent real-time transaction executions. As a result of this study, we should be able to analyze the consistency control issues and then propose algorithms

to handle the requirements of these systems.

Also, much of the current work in real-time task scheduling assumes complete knowledge of the arriving tasks along with the exact times of task arrivals and exact time of task execution. Obviously, this assumption is not always valid. The impact of the inexact knowledge on the performance of a system is not obvious. In this context, we propose to investigate these effects, and if necessary propose new scheduling algorithms to handle the practical loads.

The evaluation criteria of distributed real-time systems will be investigated and the performance of the new real-time consistency control schemes will be evaluated against these performance criteria.

This document is organized as follows. Section 2 reviews the current work done in the distributed system and real-time system areas. In Section 3 we outline the work that we propose to do in distributed real-time systems. Finally, Section 4 has some conclusions.

2 Previous Work

In this section, we summarize some of the existing work dealing with consistency control of distributed systems and scheduling in real-time systems.

2.1 Mutual Exclusion in Distributed Systems

The problem of mutual exclusion arises when several processes, operating in parallel, compete for resources that cannot be shared and therefore constraints have to be applied to ensure that when one process is using such a resource none of the others can gain access to it. There are numerous solutions for the mutual exclusion problem in distributed systems(e.g.,

[22, 26, 34, 2, 29, 8, 7, 35, 39, 40]). These fall into one of the following two categories: token based and non-token based.

In a token-based scheme, a logical token exists for each sharable resource. The process or node currently holding the token is authorized to access the corresponding resource in a mutually exclusive fashion. The simplest token-based scheme assumes a logical ring of processes or nodes along which the token is propagated. Several other complex token based schemes also exist (e.g., [29, 8, 7]). Here we briefly summarize three such schemes.

Ye-In Chang et al have proposed two variants of token-based mutual exclusion algorithms. In the first variant [7], they proposed an O(log N) token-based mutual exclusion algorithm for distributed systems. Here, a logical tree is maintained in a fully connected network, and the root is the last site to get the token among the current requesting nodes when no message is in transmission. When a node invokes mutual exclusion, it sends its request to the node possibly holding the token. The request is continuously forwarded until it arrives at the root. Therefore, the number of messages to get hold of the token is proportional to the number of nodes on the path leading to the root. To speed up the search for the token, the algorithm attempts to reduce the height of the logical tree. The message complexity of the algorithm is O(log N) in light traffic, where N is the number of nodes, and is reduced to three in heavy traffic. Furthermore, the algorithm is modified to be resilient to node failures and a recovery procedure is also presented to restore a recovering site consistently into the system.

In the second one [8], a token-based mutual exclusion algorithm for distributed systems which is fault tolerant to communication link and site failures is presented. In the algorithm, the system topology is a graph such that each site has more than one path to the site holding the token. The algorithm is fault tolerant due to the fact that a site has alternative paths

to search for the token in case of communication link or site failures. Every site communicates only with its neighboring sites and holds information only about its neighbors. When a site invokes mutual exclusion, a request message is sent along the path from the requesting site to the token-holding site. The token is passed along the same path in reverse direction and as it goes, the direction of the edges is reversed so that the path always leads to the site holding the token. This algorithm is free of deadlock, starvation and fault tolerant and has a recovery procedure to restore a recovering site consistently into the system.

M. Mizuno et al proposed a token based mutual exclusion algorithm which uses data structures similar to coteries, called quorum agreements [29]. The performance of the algorithm depends upon the quorum agreements used. When a good quorum agreement is used, the overall performance of the algorithm compares favorably with the performance of other mutual exclusion algorithms.

In a non-token based system, mutual exclusion is achieved by message communication between processes. These processes have only local variables, and the only way they can exchange information with each other is through explicit communication. There are several algorithms under this category (e.g., [22, 26, 34, 2, 35, 39, 40]). Here we describe two efficient algorithms.

In [39], Mukesh Singhal states that to guarantee mutual exclusion, the minimal connectivity needs to be maintained at all times so that when a site enters into competition for the critical section (CS), it comes to know of all the sites and only the sites which are concurrently competing for CS. To be specific, there is a request graph for the whole system which records the directions of mutual exclusion requests from sites. A directed edge from S_i to S_j denotes that whenever S_i invokes mutual exclusion, it will request permission from S_j. There should always exist a directed edge between every

pair of sites and the direction of edges is always towards the site which executed CS later with the edge adding and deleting rules.

The message traffic generated by the algorithm per CS execution has been analyzed. When the rate of CS request is low, $(n-1)$ messages are exchanged per CS execution and when the rate of CS request is high, on the average $3*(n-1)/2$ messages are exchanged per CS execution.

Solutions to mutual exclusion problem are often vulnerable to site and communication failures. Intersecting quorums can be used to provide fault-tolerant solutions, but they usually incur high communication costs. Agrawal and Abbadi present a new quorum-based algorithm which has low communication cost and can handle both types of failures [2]. Given a set of sites, we can logically organize these sites to form a tree. A quorum can be constructed by selecting any path starting from the root and ending with any of the leaves. If successful, this set of sites constitutes a quorum. If it fails to find a path as a result of the failure of a site, then the algorithm must substitute for that site with two paths, both of which starting from the children of this failed site and terminate with leaves. In this way, we can construct a tree quorum. The tree quorums formed this way satisfy the intersection and the minimality properties of coteries. In the best case when the system is free from failure, only $\lceil \log N \rceil$ sites are necessary to form a tree quorum. In the worst case $\lceil (N+1)/2 \rceil$ sites are required to form a quorum. The algorithm can tolerate the failure of up to $N - \lceil \log N \rceil$ specific sites and still form a tree quorum.

This algorithm exhibits the useful property of graceful degradation, i.e. as failures occur, and increase, the cost of forming a quorum may increase and the probability of forming a quorum decreases. The penalty for failures closer to the root is more severe than the failures in the vicinity of the leaves. The availability of an algorithm is defined as the probability of

forming a quorum successfully in that algorithm. Analysis results show that the tree quorum algorithm can achieve comparable degree of availability as the majority quorum algorithm but at substantially lower costs. In practice, this algorithm can use a spanning tree in a network. A spanning tree with a minimum radius is most appropriate for our algorithm and will result in minimum sized quorums.

None of these algorithms are suitable for use in distributed real-time systems due to lack of time guarantees. The extensions that may yield these guarantees are not obvious. We propose to look at this aspect in the proposed research.

2.2 Concurrency Control in Distributed Systems

In a typical distributed system, several transactions may be executing simultaneously. If interleaving of these transactions is not controlled, it could result in incorrect execution of transactions, and thereby an inconsistent database. This is the concurrency control problem.

To solve the concurrency control problem, concurrency control algorithms are developed. Based on implementation mechanisms, concurrency control algorithms may be classified as lock-based and timestamp-based. Similarly, based on conflict handling mechanisms, they may be classified as conservative and optimistic.

The basic idea of conservative locking is that whenever a transaction wishes to access a data item, it should first obtain an appropriate lock. If a transaction finds that a data item that it wishes to lock is already locked by another transaction, it must wait until this lock is released. When a transaction has obtained all the required locks, it proceeds with its execution.

In conservative timestamping, each transaction has a unique timestamp and data items are timestamped each time they are accessed. A transaction's

request to write a data item is valid only if that data item was last read and written by an older transaction. Similarly, A transaction's request to read a data item is valid only if that data item was last written by an older transaction. Transactions are aborted and restarted when an operation on a particular item cannot be validated.

In optimistic concurrency control it is hoped that no conflicts of access will occur. Transactions proceed until they are ready to commit, at which time a validity check is made. If conflicts with earlier transactions are detected, the transaction is aborted and must be restarted.

The performance of the above mentioned concurrency control schemes depends on the scenario. For example, optimistic concurrency control will outperform conservative schemes when the likelihood of conflict between two transactions is low. But it will be inferior when the degree of conflicts on accessing data items is high. The performance of these schemes under time-bound conditions in real-time systems is not at all clear. In fact, we first need to design algorithms that incorporate transaction timebounds into the concurrency control algorithms.

In the context of priority-based systems, priority inversion is said to have occurred when a higher priority task must wait for a lower priority task to release a shared resource. It can cause unbounded delay to high priority tasks and these tasks may miss their deadlines, thus degrading the performance of real-time systems. Recently, Liu Sha et al have suggested a concurrency control algorithm for distributed real-time systems to solve this problem[37]. This paper presents a priority-driven two-phase lock protocol called the read/write priority ceiling protocol. This algorithm ensures that a high-priority transaction can be blocked by lower priority transactions for at most the duration of a single embedded transaction. Salient features of this protocol are as follows:

- Task τ, the highest priority task ready to run, is assigned the processor. Before an embedded transaction of τ can read (or write) data object O, it must first obtain the read(or write) lock on the data object O. In addition, each embedded transaction follows the two-phase lock protocol and all the locks it holds will be released at the end of the transaction.

- Let O be the data object with the highest r/w priority ceiling of all data objects currently locked by transactions other than those of τ. When task τ's transaction attempts to lock a data object O, τ will be blocked and the lock on an object O will be denied, if the priority of task τ is not higher than the r/w priority ceiling of data object O. In this case, task τ is said to be blocked by the task whose transaction holds the lock on O. If task τ's priority is higher than the r/w priority ceiling of O, then τ is granted the lock on O.

- A task τ and its transaction T use the priority assigned to τ, unless T blocks higher priority transactions. If transaction T blocks higher priority tasks, T inherits P_H, the highest priority of the tasks blocked by T. Priority inheritance is transitive. Finally, the operations of priority inheritance and of the resumption of original priority must be indivisible.

- When a task τ does not attempt to execute an embedded transaction, it can always preempt other tasks and their embedded transactions executing at a lower priority level. This reduces the blocking time of a higher priority task from the entire execution times of lower priority tasks to only the duration of lower priority tasks' embedded transactions.

Unlike real-time systems where task conflict over shared data may last for a short period of time, in real-time database systems, transaction conflict

over shared data may last as long as the execution time of a transaction. In this case, the above algorithm may not be such a satisfactory scheme for the distributed real-time database system, plus that it is an extreme conservative one with respect to the number of transactions that can execute concurrently. Also it requires prior knowledge about the data objects to be accessed by each transaction.

Priority abort algorithms are introduced to overcome the "life-time blocking" problems of priority ceiling algorithms by aborting the low priority transaction. These priority abort schemes again have their problems that it may lead to high transaction abort rate due to data access conflict. J. Huang, J. Stankovic et al propose a conditional priority inheritance scheme [17]. Here if the low priority transaction is near completion, it inherits the priority of the high priority transaction, thus avoiding an abort with its waste of resources; otherwise, the low priority transaction is aborted, thereby avoiding the long blocking time for the high priority transaction, and also reducing the amount of wasted resources used thus far by the low priority transaction. The performance studies indicate that with respect to deadline guarantee ratio, the conditional priority inheritance scheme works better than the priority ceiling scheme as well as the priority abort schemes.

In the case of distributed systems, where resources can be distributed among the nodes connected by networks and replicate copies exist, it will become even more difficult to design a concurrency control algorithm to satisfy the time restrictions. We propose to look at this issue.

2.3 Replica Control

A replicated object is a data item that is stored redundantly at multiple locations. Replication is introduced into distributed systems to improve the reliability and availability. In addition, replication can enhance performance

by allowing user requests initiated at sites where the data are stored to be processed locally without incurring communication delays, and by distributing the workload of user request to several sites where the subtasks of a user request can be processed concurrently. These benefits of replication must be seen in the light of the additional cost and complexities introduced by replication control. Replica control algorithms are mechanisms to manage a physically distributed collection of data objects to appear as if it were a single, highly available data object. The following are some of the existing algorithms(e.g. [16, 3, 4, 14, 15, 19, 25, 33, 21, 41, 42, 20, 31, 45, 11, 43]) in this area.

Existing algorithms for update synchronization in replicated database system follow a semidistributed model of update execution because only one site completely executes an update and other sites just commit its writes. In [41], M. Singhal presents a fully-distributed approach to update synchronization where each site completely executes every update. This can certainly improve the performance by its parallelism. Slow machines will not slow down the whole system. Also it reduces the communication overhead.

In replicated database consistency may have two meaning: internal consistency and mutual consistency. Internal consistency deals with the semantics of data objects within a single database copy; mutual consistency requires that all database copies have the same value at one time. [41] deals with internally consistent state. I.e. when all update activity ceases, all copies of the replicated database must reach the same value. Since each site completely executes every update, a write only has to lock one copy. This one copy lock for write results in shorter response time and shorter lock holding time. There will be less interaction among sites, so less time wasted on synchronization. Reliability and availability are improved. And one site's failure will not affect the whole system since it will not block any operations.

The standard quorum consensus method requires a majority of sites to be involved in write operations. It can turn out to be very expensive when the number of sites is very large. A. Kumar presents a method which organizes a group of objects into a multi-level hierarchy[20]. At each level it applys the same rules of quorum consensus algorithm, i.e. a majority of sites have to participate in the operation of writes and the number of sites involved in reads and writes to the same object has to exceed the total number of sites. It will greatly reduce the number of sites involved in operations. Therefore it will definitely reduce the average message cost. A performance comparison, in terms of availability and message cost against majority voting and dynamic voting is carried out. It shows an improvement in the message cost, but no single method was found to dominate in terms of availability.

In [33], Pu, Leff et al introduce the idea of valued redundancy. It says that any object can have a value for its redundancy. By replicating only the most valuable objects, we can improve the performance of the distributed systems. The value of the system is determined by its usefulness and maintenance cost. There is a difference between weighted voting and valued redundancy: Weighted voting is an algorithm to maintain consistency among the copies of an object. Valued redundancy is an approach to manage the degree of redundancy for replicated data. Weights are assigned to the copies in weighted voting to represent the static difference relative to each other. A copy residing on a faster machine would receive a heavier weight. Values are assigned to the copies to represent the dynamic properties relative to other resources in the system, so a higher value for a copy would keep it longer in the local cache. An object with low frequence of write, high frequence of read will have higher value. That is to say, we should have more copies of this object to keep high availability.

Replication is the key factor in making distributed real-time systems more

reliable than centralized ones. However, if replication is used without proper synchronization mechanisms, consistency of the system might be violated. In [43], S.H.Son presents a synchronization algorithm for distributed real-time systems with replicated data. It reduces the time required to execute physical write operations when updates are to be made on replicated data objects, by relaxing the level of synchronization between write operations on data objects and physical write operations on copies of them. At the same time, the consistency of replicated data is not violated, and the atomicity of transactions is maintained. The algorithm exploits the multiple versions of a data object and the semantic information of read-only transactions is achieving improved system performance. The algorithm also extends the notion of primary copies such that an update transaction can be executed provided at least on token copy of each data object in the write set is available. The number of tokens for each data object can be used as a tuning parameter to adjust the rebustness of the system. Multiple versions are maintained only at the read-only copy sites, hence the storage requirement is reduced.

Reliability does not come for free. There is a cost associated with the replication of data: storage requirement and complicated control in synchronization. For appropriate mangement of multiple versions, some communication cost is inevitable to inform data objects about activities of read-only transactions. There is also a cost associated with maintaining the data structures for keeping track of versions and time-stamp.

Replication is introduced into distributed systems to improve availability which will result in fault-tolerance and high concurrency. These may be achieved at the cost of deteriorated performance in the sense of response time of update operations. How to achieve all these at the lowest cost of deteriorated performance is our research goal in the area of replica control.

2.4 Centralized and Distributed Real-Time Systems

A real-time system is a system whose correctness depends not only on the logical results of computation, but also on the time at which the results are produced. The activities in the real-time systems have timing constraints associated with them which must be satisfied in order for the system to exhibit correct behavior.

Computers have been used for real-time applications for a long time. The design and implementation of such systems has usually been carried out as an extension of the system design principles used for general purpose computer systems. In particular, real-time systems have often been designed as interrupt-driven systems with priority-based scheduling. Priority structures are used to accomplish the real-time processing by assigning higher priority to critical tasks.

Hard real-time applications are those in which the time constraints of the processing requests play a major role; that is, not meeting the constraints of an accepted processing request is considered a failure. A soft constraints on the other hand may be desirable to meet, but failure to do so does not cause a system failure. Our goal is to study the performance of distributed system in soft real-time applications.

Zhao's paper of Scheduling Tasks with Resource Requirements in Hard Real-Time System [32] proposes an algorithm with heuristic function to schedule the arriving tasks with time restrictions and resources requirements. As soon as a new task enters the system, the scheduler is going to see if it can satisfy the requirement of the new task while still satisfy the requirements of the existing tasks. So a task will either be executed by its deadline or aborted.

There are many real-time tasks scheduling algorithms like the above one in centralized systems(e.g. [23, 28, 24, 49]). Tasks scheduling in distributed

systems will be a quite different problem. Here there is no centralized mechanism which can play this role of a centralized tasks scheduler. Instead, every distributed process work on its own and they have to come to a consensus by working separately. [47, 44, 32, 46] are some of the distributed real-time tasks scheduling algorithms. Most of the existing algorithms in distributed real-time systems are concerned with the operating system level, or the allocation of resources[47], or the optimal distribution of tasks among the nodes of networks according to the availabilities of the resources on the nodes, or the workload reconfiguration in system node failure [18, 46] or the real-time communication [12, 1].

For example, in [47], H. F. Wedde, G. S. Alijani et al analyze current and future needs in designing and implementing adaptive distributed real-time systems and formulated design requirements in order to realize a highly integrated design:

- multiplicity and multifunctionality on the hardware/interconnection level;
- distributed resource allocation without any form of centralized control function on the operating system level;
- flexible and efficient distribution and relocation of information on the file system/application level.

On the operating system level, they use the transparent distributed resource scheduling scheme of their novel distributed operating system DRAGON SLAYER, thus satisfying the service requirements formulated for the integrated system design. In the MELODY file system, high reliability is achieved through replication of files. Flexibility of information handling in terms of meeting real-time constraints is achieved through the local file assigners which continuously evaluate the relevant cost and time delay factors in

cooperation with the other file assigners managing copies of this file. In this way they determine, for each file, whether the total number of copies would be increased or decreased, or where to relocate file copies to be public or private. Only public copies need to be up-to-date and consistent. Status changes would be managed by the file managers as well upon request of at least one file server.

In [46], M. Takegaki et al consider a fault-tolerant architecture for loosely coupled distributed real-time systems, and proposed a distributed task management method that aims at network-wide fault tolerance, compatible with flexible resource utilization. The task management method includes a checkpoint data transfer mechanism and an adaptive task remapping mechanism. A policy of task remapping is based on the thermodynamic diffusion model, in which tasks are diffused among sound nodes, and global balance is achieved by several repetitions of local load sharing. Convergence to this steady state is theoretically ensured. The protocol of the proposed task remapping algorithm, 2-phase diffusion algorithm, is completely distributed and weakly synchronized; each node starts the same algorithm periodically, without waiting for any messages but waits for messages from neighboring nodes during execution of the algorithm. The proposed algorithm is simple enough to execute in a real-time computing environment. Using this task remapping mechanism, we can construct a new fault tolerant system which can survive any successive nodal failures.

Many other strategies exist to solve the problem of resource allocation or the problem of tasks distribution so as to reach a good performance in the distributed real-time systems. We are more concerned with the consistency control problems. Specifically, we are concerned with adapting the existing consistency mechanisms in distributed systems to the real-time applications.

2.5 Performance Evaluations of Distributed Systems

Performance of communication systems has been well studied with both simulation and analysis methods[9, 13, 27, 6]. Performance of distributed systems which is a major issue in the design of distributed systems, however, has been studied mostly using simulations or using analysis on a simplified model[31, 45, 38, 11, 36]. That is because of the complexity of the behavior of such systems. Right now no good intuition about the performance of distributed systems is available.

Efforts have been made in building evaluation tools for testing and evaluation of distributed real-time systems [5]. Because of the complexity of the distributed real-time system, analysis and simulation techniques have become infeasible in many cases. The approach in [5] was to implement the distributed computer testbed which provides the following capabilities:

- A reconfigurable facility for implementing the relevant features of the distributed architecture under consideration, known as System Under Study(SUS).
- Comprehensive instrumentation to observe the behavior and performance of selected SUS components in real-time under a variety of conditions.

What are the performance criteria of distributed real-time systems and how to evaluate the performance of distributed real-time systems efficiently are the issues to be studied.

3 Proposed Work

As mentioned in the introduction, we are interested in the design and analysis of algorithms for distributed real-time systems. We are especially interested

in the consistency control algorithms for these systems.

In general, distributed systems are characterized by the distribution of resources. Whether the access control to the resources is centralized or distributed is still the choice of the designer. While distributed control enhances the system robustness to component failures, it has an associated cost for coordination among the control components. Since much of the work in distributed systems is aimed at building robust systems as opposed to building cost-effective and efficient systems, cost and time have been secondary objectives for designers.

Real-time systems, on the other hand, are designed with the primary objective of meeting the time-constraints of its requests. Both system cost and robustness have been the secondary objectives for the design. Accordingly, much of the effort in this area has been in developing efficient centralized control algorithms. In particular, considerable efforts are expended in the development of centralized scheduling algorithms.

However, knowing the criticality of the applications for which the real-time systems are implemented, *robustness* appears to be a necessary characteristic of these systems. Whether it is possible to effectively include both robust control and time-critical constraints into distributed real-time algorithms is the topic of investigation of this research. Especially, we are interested in the following issues.

1. Can resource replication be used to facilitate meeting the task deadlines in distributed real-time systems? In case replication is present, determine the types of replica control algorithms suitable to meet the time-critical needs as well as the consistency needs of the applications.

2. Determine the concurrency control algorithms suitable for distributed real-time systems. Especially study the characteristics of the existing concurrency control algorithms in terms of response time guaran-

tees. Also look at the possibility of designing algorithms with weaker-consistency provisions but stronger time guarantees.

3. Similar studies are necessary for designing mutual exclusion algorithms especially suitable for distributed real-time systems.

4. In addition, we propose to study other aspects of the relationships among the three important factors in these systems: time-boundedness, robustness, and consistency.

5. Evaluate the performances of these consistency mechanisms under a variety of realistic conditions.

Since real-time communication is already a well-researched area, we do not intend to invent new algorithms in the area of real-time communications. Instead, we propose to choose a set of existing algorithms with known characteristics (time-boundedness, robustness, and consistency) and pursue the above mentioned issues. Following is a detailed discussion of the issues that we propose to investigate.

3.1 Mutual Exclusion

As mentioned in Section 2.1, several solutions are proposed to solve the mutual exclusion problem in distributed systems. It is not clear if these algorithms are also applicable to distributed real-time systems. In fact, due to the absence of any guaranteed performance bounds, most of the algorithms may not be suitable for real-time applications. Hence, there is a need to either modify the existing algorithms or design new algorithms to meet the needs of the real-time systems.

For example, consider the token-ring based mutual exclusion algorithm. Here, to incorporate the task deadline concept, we can first gather the slack

time information from each of the nodes in the token ring, and then pass the token to the node with the least slack time in the preceding round. This may lower the effective utilization of the resource as well as the communication system due to the additional token rotations. But it may achieve the desired effect of meeting the deadlines of the tasks. Similar extensions are possible in non-ring based algorithms such as tree-based algorithms where priorities based on deadline can be assigned for mutually exclusive access.

We propose to investigate other mutual exclusion algorithms that are suitable for real-time systems. The proposed research will result in guidelines to design such algorithms under different system assumptions. Specifically,

- Develop criteria to evaluate the suitability of mutual exclusion algorithms to distributed real-time systems.
- Either suggest modifications to the existing algorithms or design new algorithms to meet the needs of real-time systems.
- Illustrate the evaluation criteria by applying them on a set of existing algorithms. Choose the ones that are more amenable for use in real-time systems.
- Develop guidelines for future developments.

3.2 Concurrency Control

Concurrency control is a fundamental requirement of any system that allows concurrent transactions. It enforces a type of serializability among the executing transactions. As mentioned earlier, this control can be implemented as either centralized or distributed. While centralized algorithms appear to be more suitable in guaranteeing time-bounded responses, designing such algorithms is not easy. Several centralized control algorithms (mainly in the

guise of scheduling algorithms) have already been proposed. The problem of dealing with time-bounded response becomes even more complex in the context of distributed control(communication, distributed clock, site failure, etc). While the existing algorithms have looked at properties such as deadlock, starvation, and throughputs, we need to investigate other issues such as bounded response times. Especially, the degree with which the conservative and optimistic strategies are amenable to time-boundedness needs to be determined. Similar investigations are necessary for locking and time-stamp strategies.

As an example of possible concurrency control algorithms for real-time systems, consider a conservative timestamp policy. Traditionally, the timestamps are generated based on the local clock and the site-id where the request for the resource originated. In the context of real-time systems, if it is possible to generate timestamps which also reflect the slack time of the request, then the existing concurrency control schemes can be directly used. However, any suggested timestamp generation scheme should also guarantee the imposition of required serializability. This needs further work.

We propose to research into the following issues:

- Study some practical real-time systems in terms of their concurrency control requirements, degree of transaction conflicts, serializability requirements, etc.
- Determine the suitability of centralized, semi-distributed, and distributed concurrency control in meeting the needs of real-time systems (e.g., robustness and time-boundedness).
- Determine the appropriateness of the time-stamp based and lock-based CC algorithms for real-time applications.

- Determine the suitability of optimistic and conservative algorithms to meet the needs of the systems.

- Based on the above study, develop concurrency control schemes for a chosen set of real-time system specifications

3.3 Replica Control

As mentioned in Section 2.3, replication is claimed to improve fault-tolerance and availability of distributed systems. These goals are achieved through appropriate replica control algorithms that control the number of copies to be accessed for read and write operations on resources. Clearly, the time-bound aspect of transactions (or requests) was hardly considered in the design of these algorithms.

Some of the concerns in applying the existing replica control algorithms to distributed real-time systems are:

- Will replication help in the distributed real-time systems? If yes, how can the replication in distributed systems help in distributed real-time systems?

- While the algorithms perform well under failures, the overhead is high when no failures are present. In the context of real-time systems, we need acceptable performance (time-bounded response time, for example) under all conditions.

- If the requests to local resources are more frequent than non-local accesses, the replica-control algorithm should be modified accordingly.

In the context of distributed real-time systems, we propose to investigate the following issues.

- Characterize the current replica control algorithms in terms of their suitability to real-time systems.
- Study the different consistency and time-bounds requirements in different distributed real-time applications and design suitable algorithms for typical applications.
- Suggest a few techniques to include the time-boundedness into the algorithms. These are general techniques that may need further tailoring for a given algorithm.
- Design a set of replica control algorithms based on the suggested techniques.
- Evaluate the algorithms.

3.4 Performance Evaluation

Distributed systems have focused on optimizing system performance in an average sense, i.e. minimize average response time and maximize average throughput. Real-time scheduling, on the other hand, has focused on the timing needs of individual computations. In the distributed real-time systems, the performance metrics should include the satisfactory rate of meeting deadlines, average response time and throughput.

Much of the existing algorithms in real-time systems assume complete knowledge of the transactions: arrival times, resource requirements, computation time, etc. However, many times this knowledge is available only as distributions. Depending on the variations of the distributions, some of the scheduling algorithms which are rated high for real-time systems, may in fact perform poorly under such variations. In addition, several exception handling techniques may be necessary to handle such situations. We propose

to investigate the impact of such distribution assumptions on the existing algorithms, and evaluate their sensitivity to the variations in the values. This involves developing a distributed real-time system simulator and a load generator. The work is currently in progress.

In this context, we propose to investigate the following issues:

- Determine some distributions for arrivals, resource and computation requirements of some practical distributed real-time systems.
- Evaluate the performance of the existing real-time system algorithms (e.g., scheduling algorithms) under practical loads. (This will involve simulations.)
- Investigate and develop the performance criteria for distributed real-time system.
- Evaluate the existing distributed system algorithms and newly developed distributed real-time system algorithms against the performance criteria for distributed real-time systems.
- If the current performance evaluation tools are found to be unsuitable or suboptimal, attempt to design new tools.

4 Conclusion

A distributed system is a system with a number of processing elements and storage devices, connected together by a communication network. It has been a well studied subject as far as the consistency control mechanisms are concerned. Real-time applications are those applications with strict requirements on the timing behavior of the system. Here we propose to investigate

the consistency control mechanisms of distributed system in real-time applications.

Distributed real-time systems are systems that consist of several dispersed concurrent processes that may have complex and time critical interactions. Most of the work in this area has been going from the real-time systems studies to distributed real-time systems to make use of the good features of fault-tolerance and high availability of the distributed systems. Our work is from the other direction. That is, we approach the distributed real-time systems problems from the existing distributed systems and try to design and analyze the distributed system algorithms with real-time requirements. In particular we are try to investigate the consistency control mechanisms of the distributed real-time systems.

As the distributed systems will be the future of computer systems and real-time is and will be the features required by the computer users, our research will prove to be of great significance in computer science.

References

[1] T. Ae, M. Yamashita, and H. Matsumoto. A response time estimation of real-time networks. *Proceedings real-time systems symposium*, pages 198–207, Dec.87.

[2] Divyakant Agrawal and Amr El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM transactions on computer systems*, Vol 9, No. 1:1–20, Feb.91.

[3] B. S. Bacarisse and S. Bek Baydere. Reliability of replicated files in partitioned networks. *IEEE*, pages 98–101, Mar.90.

[4] Daniel Barbara and Hector Garcia-Molina. The case for controlled inconsistency in replicated data(position paper). *IEEE*, pages 35–38, Mar.90.

[5] Devesh Bhatt, Adel Ghonami, and Ranga Ramanujan. An instrumented testbed for real-time distributed systems development. *Proceedings real-time systems symposium*, pages 241–250, Dec.87.

[6] Laxmi N. Bhuyan, Dipak Ghosal, and Qing Yang. Approximate analysis of single and multiple ring networks. *IEEE transactions on computers*, Vol. 38. No. 7:1027–1036, Jul.89.

[7] Ye-In Chang, Mukesh Singhal, and Ming T. Liu. An improved o(log n) mutual exclusion algorithm for distributed systems. *1990 International Conference on parallel processing*, pages III295–III302, 90.

[8] Ye-In Chang, Mukesh Singhal, and Ming T. Liu. A fault tolerant algorithm for distributed mutual exclusion. *9th Synposium on Reliable Distributed System*, Oct.90.

[9] Vijay Chauhan and Adarshpal S. Sethi. Performance studies of token based local area networks. *Proceddings of 10th Conference on Local Computer Networks*, pages 100–107, Oct.85.

[10] W. W. Chu and C.-M. Sit. Estimating task response time with contentions for real-time distributed systems. *Proceedings real-time systems symposium*, pages 272–281, Dec.88.

[11] Bruno Ciciani, Daniel M. Dias, and Philip S. Yu. Analysis of replication in distributed database systems.

[12] L. Ciminiera and A. Valenzano. Performance analysis of acknowledgment mechanisms in token-bus networks. *Proceedings real-time systems symposium*, pages 179–185, Dec.87.

[13] Timothy A. Gonsalves and Fouad A. Tobagi. Comparative performance of voice/data local area networks. *IEEE journal on selected areas in communications*, Vol. 7. No. 5.:657–669, Jun.89.

[14] Anna Hac. A distributed algorithm for performance improvement through file replication, file migration, and process migration. *IEEE transactions on software engineering*, Vol.15, No.11:1459–1470, Nov.89.

[15] Maurice Herlihy. Type-specific replication algorithms for multiprocessors. *IEEE*, pages 70–74, Mar.90.

[16] Hui-I Hsiao and David J. DeWitt. Replicated data management in the gamma database machine. *IEEE*, pages 79–84, Mar.90.

[17] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. On using priority inheritance in real-time databases. *Proceedings real-time systems synposium*, pages 210–221, Jul.91.

[18] R. M. Kieckhafer. Task reconfiguration in a distributed real-time system. *Proceedings real-time systems symposium*, pages 25–32, Dec.87.

[19] James Jay Kistler. Increasing file system availability through second-class replication. *IEEE*, pages 65–69, Mar.90.

[20] Akhil Kumar. Performance analysis of a hierarchical quorum consensus algorithm for replicated objects. *IEEE*, Jul.90.

[21] Rivka Ladin. Lazy replication: Exploiting the semantics of distributed services. *IEEE*, pages 31–34, Mar.90.

[22] L. Lamport. Time, clocks and ordering of events in distributed systems. *Communications of ACM*, pages 558–565, Jul.78.

[23] Shem-Tov Levi and Ashok K. Agrawala. Real-time system design, 90.

[24] Shem-Tov Levi, Daniel Mosse, and Ashok K. Agrawala. Allocation of real-time computations under fault tolerance constraints. *Proceedings real-time systems symposium*, pages 161–171, Dec.88.

[25] Darrell D. E. Long. Analysis of replication control protocols. *IEEE*, pages 117–122, Mar.90.

[26] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in dicentralized systems. *ACM transaction on computer systems*, pages 145–159, May.85.

[27] Peter Martini, Otto Spaniol, and Thomas Welzel. File transfer in high-speed token ring networks: performance evaluation by approximate analysis and simulation. *IEEE journal on selected areas in communications*, Vol. 6. No. 6:987–996, Jul.88.

[28] Frank William Miller. A predictive real-time scheduling algorithm. *Department of computer science, the university of Iowa*, Aug.89.

[29] Masaaki Mizuno, Mitchell L. Neilsen, and Raghavendra Rao. A token based distributed mutual exclusion algorithm based on quorum agreements. *IEEE*, pages 361–368, Jul.91.

[30] Sape Mullender. *Distributed systems.* ACM, 89.

[31] R. D. Nelson and B. R. Iyer. Analysis of a replicated data base. *Performance Evaluation*, pages 133–148, May.85.

[32] Francis J. Prusker, Edward Pl Wobber, Wei Zhao, Krithivasan Ramamritham, and John A. Stankovic. The siphon: managing distant replicated repositories scheduling tasks with resource requirments in hard real-time systems. *IEEE proceedings of the workshop on management of*

replicated data IEEE transactions on software engineering, Vol. SE-13, No. 5:44–47 564–573, Nov.90 May.87.

[33] Calton Pu, Avraham Leff, Frederick Korz, and Shu-Wie Chen. Valued redundancy. *IEEE*, pages 76–78, Mar.90.

[34] Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM transactions on computer systems*, pages 61–77, Feb.89.

[35] Beverly A. Sanders. The information structure of distributed mutual exclusion algorithms, Jun.86.

[36] Kenneth C. Sevcik. Comparison of concurrency control methods using analytic models. *Information Processing*, 83.

[37] Liu Sha, Ragunathan Rajkumar, Sang Hyuk Song, and Chun-Hyon Chang. A real-time locking protocol. *IEEE transaction on computers*, Vol.40, No.7:793–800, Jul.91.

[38] Shiow-Chen Shyu and Victor O. K. Li. Performance analysis of static locking in distributed database systems. *IEEE transaction on Computers*, Vol.39, No.6, Jun.90.

[39] Mukesh Singhal. A dynamic information-structure mutual exclusion algorithm for distributed systems. *IEEE ICDCS*, pages 70–78, 89.

[40] Mukesh Singhal. Theory and construction of optimal dynamic information-structure mutual exclusion algorithms for distributed systems. *IEEE transaction on parallel and distributed systems*, pages 70–78, 89.

[41] Mukesh Singhal. Update transport: a new technique for update synchronization in replicated database systems. *IEEE transactions on software engineering*, Vol.16, No.12:1325–1336, Dec.90.

[42] Mukesh Singhal and A. K. Agrawala. A concurrency control algorithm and its performance for replicated database systems. *IEEE*, pages 140–147, Sep.86.

[43] S. H. SON. Using replication for high performance database support in distributed real-time systems. *Proceedings real-time systems symposium*, pages 79–85, Dec.87.

[44] Jay K. Strosnider, Tom Marchok, and John Lehoczky. Advanced real-time scheduling using the ieee 802.5 token ring. *Proceedings real-time systems symposium*, pages 42–52, Dec.88.

[45] U. Sumita and O. R. Liu Sheng. Analysis of query processing in distributed database systems with fully replicated files: A hierarchical approach. *Performamce Evalluation*, pages 223–238, Aug.88.

[46] M. Takegaki, H. Kanamaru, and M. Fujita. The diffusion model based task remapping for distributed real-time systems. *Proceedings real-time systems synposium*, pages 2–11, Jul.91.

[47] H. F. Wedde, G. S. Alijani, F. Kang, and B.-K. Kim. Melody: a distributed real-time testbed for adaptive systems. *Proceedings real-time systems symposium*, pages 112–119, Dec.88.

[48] C. M. Woodside and D. W. Craig. Local non-preemptive scheduling policies for hard real-time distributed systems. *Proceedings real-time systems symposium*, pages 12–16, Dec.87.

C-2

[49] Wei Zhao, Krithivasan Ramamritham, and John A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE transactions on computers*, Vol. C-36, No. 8:949–960, Aug.87.

MUTUAL EXCLUSION AND REPLICA CONTROL IN DISTRIBUTED REAL-TIME SYSTEMS: A PROPOSAL

Yinghong Kuang
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

June 29, 1992

1 Introduction

A distributed system is a system with a number of processing elements and storage devices, connected together by a communication network [24]. Potentially, this makes a distributed system more powerful than a conventional, centralized one in two ways. First, it can be more reliable, because every function can be replicated several times. Second, a distributed system can do more work in the same amount of time, because many computations can be carried out in parallel. These two properties, fault tolerance and parallelism, give a distributed system the potential to be much more powerful than a centralized one.

Real-time applications are those applications with strict requirements on the timing behavior of the system. The system that support the execution of real-time applications ensuring that the timing requirements are met are often referred to as real-time systems. Task scheduling is a vital component of real-time systems. Accordingly, this is a widely researched area resulting in real-time schedulers [39, 35, 22].

Due to the potential for parallelism and fault-tolerance in distributed systems, they are increasingly the preferred architectures for real-time systems. Here, by real-time distributed systems we mean those systems that consist of several dispersed concurrent processes that communicate with one another by exchanging messages and may have complex and time-critical interactions [5]. The complex interactions include sharing tasks among the nodes of a network, scheduling of tasks and maintaining consistency among processes [38].

The existing approaches to distributed real-time systems include enhancements to operating systems [18, 22, 37], communication systems [10, 1] as well as distribution of subtasks according to resource allocation and fault tolerance requirement [19, 8, 39, 26, 37, 19]. While some studies start from the centralized implementation of real-time systems and attempt to intro-

duce the distributed processing aspects in to the systems, others are starting from the general distributed systems and attempt to consider real-time systems as a viable application. Here, we chose the latter approach.

We propose to study the consistency control aspects of distributed real-time systems. In particular, we propose to concentrate on two issues mentioned above: mutual exclusion and replica control. Since the existing solutions to these issues are mainly aimed at non-real time systems with emphasis on throughput and availability rather than on time-bounded responses, the proposed work will be useful in building distributed real-time systems. To achieve this task, we will attempt to establish a relationship between time-boundedness, availability and concurrency in these systems. This may also require formalization of the semantics of concurrent real-time transaction executions. As a result of this study, we should be able to analyze the consistency control issues and then propose algorithms to handle the requirements of these systems.

This document is organized as follows. Section 2 reviews the current work done in the areas of the mutual exclusion and replica control for distributed system and real-time systems. In Section 3 we outline the work that we propose to do in distributed real-time systems. Finally, Section 4 has some conclusions.

2 Previous Work

In this section, we summarize some of the existing work dealing with mutual exclusion and replica control of distributed systems.

2.1 Mutual Exclusion in Distributed Systems

The problem of mutual exclusion arises when several processes, operating in parallel, compete for resources that cannot be shared and therefore constraints have to be applied to ensure that when one process is using such

a resource none of the others can gain access to it. There are numerous solutions for the mutual exclusion problem in distributed systems(e.g., [17, 21, 28, 2, 23, 7, 6, 29, 30, 31]). These fall into one of the following two categories: token based and non-token based.

In a token-based scheme, a logical token exists for each sharable resource. The process or node currently holding the token is authorized to access the corresponding resource in a mutually exclusive fashion. The simplest token-based scheme assumes a logical ring of processes or nodes along which the token is propagated. Several other complex token based schemes also exist (e.g., [23, 7, 6]). Here we briefly summarize three such schemes.

Ye-In Chang et al have proposed two variants of token-based mutual exclusion algorithms. In the first variant [6], they proposed an O(log N) token-based mutual exclusion algorithm for distributed systems. Here, a logical tree is maintained in a fully connected network, and the root is the last site to get the token among the current requesting nodes when no message is in transmission. When a node invokes mutual exclusion, it sends its request to the node possibly holding the token. The request is continuously forwarded until it arrives at the root. Therefore, the number of messages to get hold of the token is proportional to the number of nodes on the path leading to the root. To speed up the search for the token, the algorithm attempts to reduce the height of the logical tree. The message complexity of the algorithm is O(log N) in light traffic, where N is the number of nodes, and is reduced to three in heavy traffic. Furthermore, the algorithm is modified to be resilient to node failures and a recovery procedure is also presented to restore a recovering site consistently into the system.

In the second one [7], a token-based mutual exclusion algorithm for distributed systems which is fault tolerant to communication link and site failures is presented. In the algorithm, the system topology is a graph such that each site has more than one path to the site holding the token. The algorithm is fault tolerant due to the fact that a site has alternative paths

to search for the token in case of communication link or site failures. Every site communicates only with its neighboring sites and holds information only about its neighbors. When a site invokes mutual exclusion, a request message is sent along a path from the requesting site to the token-holding site. The token is passed along the same path in reverse direction and as it goes, the direction of the edges is reversed so that the path always leads to the site holding the token. This algorithm is free of deadlock, starvation and fault tolerant and has a recovery procedure to restore a recovering site consistently into the system.

M. Mizuno et al proposed a token based mutual exclusion algorithm which uses data structures similar to coteries, called quorum agreements [23]. The performance of the algorithm depends upon the quorum agreements used. When a good quorum agreement is used, the overall performance of the algorithm compares favorably with the performance of other mutual exclusion algorithms.

In a non-token based system, mutual exclusion is achieved by message communication between processes. These processes have only local variables, and the only way they can exchange information with each other is through explicit communication. There are several algorithms under this category (e.g., [17, 21, 28, 2, 29, 30, 31]). Here we describe two efficient algorithms.

In [30], Mukesh Singhal states that to guarantee mutual exclusion, the minimal connectivity needs to be maintained at all times so that when a site enters into competition for the critical section (CS), it comes to know of all the sites and only the sites which are concurrently competing for CS. To be specific, there is a request graph for the whole system which records the directions of mutual exclusion requests from sites. A directed edge from S_i to S_j denotes that whenever S_i invokes mutual exclusion, it will request permission from S_j. There should always exist a directed edge between every pair of sites and the direction of edges is always towards the site

which executed CS later with the edge adding and deleting rules.

The message traffic generated by the algorithm per CS execution has been analyzed. When the rate of CS request is low, $(n-1)$ messages are exchanged per CS execution and when the rate of CS request is high, on the average $3*(n-1)/2$ messages are exchanged per CS execution.

Solutions to mutual exclusion problem are often vulnerable to site and communication failures. Intersecting quorums can be used to provide fault-tolerant solutions, but they usually incur high communication costs. Agrawal and Abbadi present a new quorum-based algorithm which has low communication cost and can handle both types of failures [2]. Given a set of sites, we can logically organize these sites to form a tree. A quorum can be constructed by selecting any path starting from the root and ending with any of the leaves. If successful, this set of sites constitutes a quorum. If it fails to find a path as a result of the failure of a site, then the algorithm must substitute for that site with two paths, both of which starting from the children of this failed site and terminate with leaves. In this way, we can construct a tree quorum. The tree quorums formed this way satisfy the intersection and the minimality properties of coteries. In the best case when the system is free from failure, only $\lceil \log N \rceil$ sites are necessary to form a tree quorum. In the worst case $\lceil (N+1)/2 \rceil$ sites are required to form a quorum. The algorithm can tolerate the failure of up to $N - \lceil \log N \rceil$ specific sites and still form a tree quorum.

This algorithm exhibits the useful property of graceful degradation, i.e. as failures occur, and increase, the cost of forming a quorum may increase and the probability of forming a quorum decreases. The penalty for failures closer to the root is more severe than the failures in the vicinity of the leaves. The availability of an algorithm is defined as the probability of forming a quorum successfully in that algorithm. Analysis results show that the tree quorum algorithm can achieve comparable degree of availability as the majority quorum algorithm but at substantially lower costs. In practice,

this algorithm can use a spanning tree in a network. A spanning tree with a minimum radius is most appropriate for our algorithm and will result in minimum sized quorums.

None of these algorithms are suitable for use in distributed real-time systems due to lack of time guarantees. The extensions that may yield these guarantees are not obvious. We propose to look at this aspect in the proposed research.

2.2 Replica Control

A replicated object is a data item that is stored redundantly at multiple locations. Replication is introduced into distributed systems to improve the reliability and availability. In addition, replication can enhance performance by allowing user requests initiated at sites where the data are stored to be processed locally without incurring communication delays, and by distributing the workload of user request to several sites where the subtasks of a user request can be processed concurrently. These benefits of replication must be seen in the light of the additional cost and complexities introduced by replication control. Replica control algorithms are mechanisms to manage a physically distributed collection of data objects to appear as if it were a single, highly available data object. The following are some of the existing algorithms(e.g. [13, 3, 4, 11, 12, 14, 20, 27, 16, 32, 33, 15, 25, 36, 9, 34]) in this area.

Existing algorithms for update synchronization in replicated database system follow a semidistributed model of update execution because only one site completely executes an update and other sites just commit its writes. In [32], M. Singhal presents a fully-distributed approach to update synchronization where each site completely executes every update. This can certainly improve the performance by its parallelism. Slow machines will not slow down the whole system. Also it reduces the communication overhead.

In replicated database consistency may have two meaning: internal consistency and mutual consistency. Internal consistency deals with the semantics of data objects within a single database copy; mutual consistency requires that all database copies have the same value at one time. [32] deals with internally consistent state. I.e. when all update activity ceases, all copies of the replicated database must reach the same value. Since each site completely executes every update, a write only has to lock one copy. This one copy lock for write results in shorter response time and shorter lock holding time. There will be less interaction among sites, so less time wasted on synchronization. Reliability and availability are improved. And one site's failure will not affect the whole system since it will not block any operations.

The standard quorum consensus method requires a majority of sites to be involved in write operations. It can turn out to be very expensive when the number of sites is very large. A. Kumar presents a method which organizes a group of objects into a multi-level hierarchy[15]. At each level it applys the same rules of quorum consensus algorithm, i.e. a majority of sites have to participate in the operation of writes and the number of sites involved in reads and writes to the same object has to exceed the total number of sites. It will greatly reduce the number of sites involved in operations. Therefore it will definitely reduce the average message cost. A performance comparison, in terms of availability and message cost against majority voting and dynamic voting is carried out. It shows an improvement in the message cost, but no single method was found to dominate in terms of availability.

In [27], Pu, Leff et al introduce the idea of valued redundancy. It says that any object can have a value for its redundancy. By replicating only the most valuable objects, we can improve the performance of the distributed systems. The value of the system is determined by its usefulness and maintenance cost. There is a difference between weighted voting and valued re-

dundancy: Weighted voting is an algorithm to maintain consistency among the copies of an object. Valued redundancy is an approach to manage the degree of redundancy for replicated data. Weights are assigned to the copies in weighted voting to represent the static difference relative to each other. A copy residing on a faster machine would receive a heavier weight. Values are assigned to the copies to represent the dynamic properties relative to other resources in the system, so a higher value for a copy would keep it longer in the local cache. An object with low frequence of write, high frequence of read will have higher value. That is to say, we should have more copies of this object to keep high availability.

Replication is the key factor in making distributed real-time systems more reliable than centralized ones. However, if replication is used without proper synchronization mechanisms, consistency of the system might be violated. In [34], S.H.Son presents a synchronization algorithm for distributed real-time systems with replicated data. It reduces the time required to execute physical write operations when updates are to be made on replicated data objects, by relaxing the level of synchronization between write operations on data objects and physical write operations on copies of them. At the same time, the consistency of replicated data is not violated, and the atomicity of transactions is maintained. The algorithm exploits the multiple versions of a data object and the semantic information of read-only transactions is achieving improved system performance. The algorithm also extends the notion of primary copies such that an update transaction can be executed provided at least on token copy of each data object in the write set is available. The number of tokens for each data object can be used as a tuning parameter to adjust the rebustness of the system. Multiple versions are maintained only at the read-only copy sites, hence the storage requirement is reduced.

Reliability does not come for free. There is a cost associated with the replication of data: storage requirement and complicated control in synchro-

nization. For appropriate m ogement3:f multiple versions, some communication cost is inevitable to inform data objects about activities of read-only transactions. There is also a cost associated with maintaining the data structures for keeping track of versions and time-stamp.

Replication is introduced into distributed systems to improve availability which will result in fault-tolerance and high concurrency. These may be achieved at the cost of deteriorated performance in the sense of response time of update operations. How to achieve all these at the lowest cost of deteriorated performance is our research goal in the area of replica control.

3 Proposed Work

As mentioned in the introduction, we are interested in the design and analysis of algorithms for distributed real-time systems. We are especially interested in the consistency control algorithms for these systems.

In general, distributed systems are characterized by the distribution of resources. Whether the access control to the resources is centralized or distributed is still the choice of the designer. While distributed control enhances the system robustness to component failures, it has an associated cost for coordination among the control components. Since much of the work in distributed systems is aimed at building robust systems as opposed to building cost-effective and efficient systems, cost and time have been secondary objectives for designers.

Real-time systems, on the other hand, are designed with the primary objective of meeting the time-constraints of its requests. Both system cost and robustness have been the secondary objectives for the design. Accordingly, much of the effort in this area has been in developing efficient centralized control algorithms. In particular, considerable efforts are expended in the development of centralized scheduling algorithms.

However, knowing the criticality of the applications for which the real-

time systems are implemented, *robustness* appears to be a necessary characteristic of these systems. Whether it is possible to effectively include both robust control and time-critical constraints into distributed real-time algorithms is the topic of investigation of this research. Especially, we are interested in the following three issues.

1. Can resource replication be used to facilitate meeting the task deadlines in distributed real-time systems? In case replication is present, determine the types of replica control algorithms suitable to meet the time-critical needs as well as the consistency needs of the applications.

2. Similar studies are necessary for designing mutual exclusion algorithms especially suitable for distributed real-time systems. Investigate the suitability of the existing distributed mutual exclusion algorithms for real-time applications. If they are found to be unsuitable, identify the inherent characteristics of the applications and those of the mutual exclusion problems that contribute to the problem. Using this information, state the trade-off relationship between the application requirements and those of the distributed mutual exclusion solutions. This study could lead to more application specific solutions to mutual exclusion in distributed real-time systems.

3. In addition, we propose to study other aspects of the relationships among the three important factors in these systems: time-boundedness, robustness, and consistency. (Note: Answers to this issue will be found as we proceed with research on issues 1 and 2.)

Since real-time communication is already a well-researched area, we do not intend to invent new algorithms in the area of real-time communications. Instead, we propose to choose a set of existing algorithms with known characteristics (time-boundedness, robustness, and consistency) and pursue the above mentioned issues. Following is a detailed discussion of the issues that

we propose to investigate.

3.1 Mutual Exclusion

3.1.1 Problem Statement

As mentioned in Section 2.1, several solutions are proposed to solve the mutual exclusion problem in distributed systems. It is not clear if these algorithms are also applicable to distributed real-time systems. In fact, due to the absence of any guaranteed performance bounds, most of the algorithms may not be suitable for real-time applications. When we consider solutions for mutual exclusion problems in general distributed systems, we only consider the performance criteria such as average response time, average throughput, and resource utilization. But in real-time applications, meeting time-constraints is considered more important than anything else. Here it is desirable to perform tasks as fast as possible, but it is more important that they meet their deadlines. So how to adapt the existing consistency control schemes to real-time applications, how to develop new schemes, and how to evaluate these schemes in the environment of real-time applications are the goals of this research. Specifically, we will concentrate on the following issues:

- **Develop criteria to classify/evaluate mutual exclusion algorithms of distributed real-time systems.** This work should result in metrics to express the suitability of a given ME algorithm to distributed real-time applications. Even though the optimistic algorithms appear to be more suitable to real-time applications than the conservative ones, it is not clear if this classification is sufficient.

- **Suggest modifications to existing mutual exclusion algorithms to meet the needs of real-time applications.** Having arrived at a classification, we propose to analyze some of the existing ME al-

gorithms and classify them accordingly. In addition, we propose to suggest modifications to the algorithms to transfer them from a less suitable class to a more desirable class. This should be possible by changing the grant/release rules in an algorithm.

- **Develop new mutual exclusion algorithms for distributed real-time systems.** Using the properties derived from the classification, we will attempt to construct new algorithms that are suitable for real-time applications. In fact, it may result in a suite of algorithms where the choice will depend on the semantics of the application.

- **Evaluate the algorithms using the criteria developed above.** This may involve using both analytical and simulation tools.

- **Develop guidelines for future development.** If in the process of development and analysis, we have developed sufficient insight regarding the applications and the algorithms, we may be able to develop some general guidelines for future work. However, this should be perceived more as wishful thinking than as a promised delivery.

3.1.2 Research Outline

First of all, we will study the existing mutual exclusion algorithms. We will examine their suitability in the environment of real-time applications. Different real-time applications may have different conditions and requirements. For example, in some applications, the load at each node is uniform, while in other applications load vary from node to node to a great extent. In some applications, the time constraints may be hard, while in others it may be soft. These are the different requirement put on the systems. Some of the algorithms may suit better in applications with certain kind of conditions and requirements while others may suit better in other applications.

Second, we should look into the possibilities of making minor modifications to the existing mutual exclusion algorithms to adapt them to the real-time applications. For example, if we take Lamport's mutual exclusion algorithm, mutual exclusion requests are served on the basis of FCFS according to the logical timestamps. If in real-time applications, we will server these mutual exclusion requests according to their slack times, it may solve the mutual exclusion problem of soft real-time applications.

Thirdly, we can also develop new algorithms just for real-time applications. Here we will determine the application's circumstances and requirements and build new schemes which will produce favorable performance results in these applications.

Fourth, we will examine the fault-tolerance feature of these mutual exclusion algorithms for distributed real-time systems.

3.1.3 Evaluations

We will evaluate the mutual exclusion algorithms under some chosen circumstances:

- Selection of the underlying architecture: We will mainly consider the bus structure of local area networks. In addition, we will consider pint-to-point linked networks. In choosing the appropriate bus structure, we need to consider factors such as (1) whether or not the bus conforms to a standard specification; (2) the expected future compatibility for an expanded system; (3) the arbitration method used in minimizing contention for the bus; (4) the total bandwidth of the bus; In addition
- The nature of arrivals of mutual exclusion requests,
- Distribution of mutual exclusion requests among processes, and
- Distribution of execution times of these requests.

According to the application requirements, we will develop criteria for these distributed real-time systems. The possible criteria may include:

- The successful rate of meeting deadlines: what is the rate of tasks with time constraints meeting their deadlines?
- If it is a soft real-time application, how will we evaluate the satisfaction rate of time constraints in practical situation? What will be an appropriate measure for the performance?
- Throughput: in mutual exclusion throughput may be, to some extent, determined by the response time. The faster these mutual requests are processed, the higher the throughput.
- The utilization: what will be the utilization of the resources?
- The overhead: what will be the amount of traffic caused by these algorithms? What will be the complexity of these algorithms, i.e. how much of the CPU time will they take?

3.1.4 How to Achieve It?

In the past, we have built analytical and simulation models for typical logical token ring mutual exclusion algorithm and have got some result of the performance of distributed systems such as response time, token rotation time, throughput, and resource utilization. We can extend these studies to measure the time boundedness of the systems. We propose to carry out similar performance studies with different underlying architectures, reliability of resources, etc.

Lastly, the result will be explained, conclusions will be drawn, experience will be gathered, which may be of great help for the studies of replica control problem in real-time applications.

3.2 Replica Control

3.2.1 Problem Statement

As mentioned in Section 2.3, replication is claimed to improve fault-tolerance and availability of distributed systems. These goals are achieved through appropriate replica control algorithms that control the number of copies to be accessed for read and write operations on resources. Clearly, the time-bound aspect of transactions (or requests) was hardly considered in the design of these algorithms.

Some of the concerns in applying the existing replica control algorithms to distributed real-time systems are:

- Will replication help in the distributed real-time systems? If yes, how can the replication in distributed systems help in distributed real-time systems?
- While the algorithms perform well under failures, the overhead is high when no failures are present. In the context of real-time systems, we need acceptable performance (time-bounded response time, for example) under all conditions.
- If the requests to local resources are more frequent than non-local accesses, the replica-control algorithm should be modified accordingly.

In particular, we will concentrate on the following issues:

- Model some typical application related semantics where replication is needed to improve the reliability and availability.
- Develop criteria to evaluate the replica control algorithms of distributed real-time systems.
- Study the existing replica control algorithms in the environment of real-time applications. Characterize them in terms of their suitability to real-time systems.

- Develop new replica control algorithms for distributed real-time systems.
- Evaluate the algorithms with the established criteria.

3.2.2 Research Outline

First of all, we will study the existing replica control algorithms. We will examine their suitability in the environment of real-time applications. Different real-time applications may have different conditions and requirements. Some of the algorithms may suit better in applications with certain kind of conditions and requirements while others may suit better in other applications.

For example, the semantics of the applications may vary. In some applications, a weaker ordering like a causal order will be required; In others, a stronger ordering like non-deterministic total order may be required; Still in others, a total order may be needed. With different requirements, we may apply different replica control mechanisms to improve the response time and availability.

Second, we should analyze the existing replica control algorithms to see their suitability to the real-time applications. By suitability, we mean their satisfaction to certain criteria, which we will mention in the following section. But since requirement of applications may vary, criteria for the applications may be different. This work will involve some simulation and analysis.

Thirdly, we should develop new algorithms for real-time applications. Here we will determine the application's circumstances and requirements and develop new schemes which will produce favorable performance results in these applications. The good features of distributed systems that will be of benefit to real-time applications are high availability and reliability, which can be achieved by redundant resources. So some possible potentials in developing new algorithm for replica control are keeping redundant resources

and multiple versions. It will be at the cost of more storage needed and more execution time. How to develop new algorithms with good performance with low cost is what we are going to work on in this research.

Fourth, we will examine the fault-tolerance feature of these replica control algorithms for distributed real-time systems. The primary mechanism for achieving the required degree of reliability in real-time systems is the use of redundancy. In a monolithic system, this means having a complete spare computer that is switched in when a failure is detected. A distributed system is easier to design for fault tolerance and is usually more cost effective than a monolithic system. Redundant hardware elements in a distributed system can be limited to replacing only a fraction of the total number of processors and certain of the other elements, but not the complete system. Software redundancy is also part of the overall design strategy to achieve fault tolerance. The degree of fault tolerance built into a system can range from an almost 100 percent reliability with standby auto-switchable spares, to a state of graceful degradation where as much backup and recovery as possible is performed before the system is brought down for repair or bug fixes. We will investigate the system degradation under faulty situations.

3.2.3 Evaluation

We will evaluate the replica control algorithms under some chosen circumstances:

- What will be the underlying architecture?
- What will be the semantic information of a chosen application?
- What will be the data distribution on the distributed systems?
- What will be the locality of data access of transactions?
- What will be the reliability of these data resources?

- Should we have more redundancy for the more critical data objects?
- What will be the distribution of transaction execution time of these requests?

These are the conditions we have to consider when we are building the model for performance analysis.

According to the application requirements, we will develop criteria for these distributed real-time systems. The possible criteria may include:

- Under what situation will replication help in real-time distributed systems? In what ways?
- For a chosen model, what will be the good degree of replication? There will be a trade-off between availability and complexity of operations.
- The successful rate of meeting deadlines: what is the rate of tasks with time constraints meeting their deadlines?
- If it is a soft real-time application, how will we evaluate the satisfaction rate of time constraints in practical situation? What will be an appropriate measure for the performance?
- Throughput: what will be the throughput of the system?
- The utilization: what will be the utilization of the resources?
- The overhead: what will be the amount of traffic caused by these algorithms? What will be the complexity of these algorithms, i.e. how much of the CPU time will they take?

3.2.4 How to Achieve It?

First we will build models for typical semantic applications with parameters such as the underlying architecture, data distribution, reliability of

resources, locality of data access, redundancy of resources, and transaction execution time distribution. Because of the complexity of the systems, simulation will be the main tool used to analyze these models. When it becomes difficult to evaluate the performance with reasonable amount of simulation time, we may resort to approximate analytical model to help simulation. The details will be clear as we proceed along with the research.

Appropriate values will be chosen for these parameters to run the simulation. The impact of the parameters will be observed, and we will try to run the simulation only with the variation of important parameters.

Most importantly, we will see the impact of replication to the performance of the distributed real-time systems. The performance criteria include the successful rate of meeting deadlines, the satisfaction of real-time constraints according to the application, the throughput, the resource utilization and the overhead of implementing the tested mechanism.

The result will be explained, which may again help developing good algorithms for distributed real-time systems.

4 Plan

We propose to complete the research in the next 18 months. Following is a detailed plan of activities.

1. **May'92 – August'92:** Survey the requirements of some typical distributed real-time systems and build models for these systems.

2. **August'92 – October'92:** Develop criteria to evaluate the performance of distributed real-time systems. The selection of criteria should take into account their usability in design and analysis of these systems as well as the ease with which they could be measured.

3. **October'92 – December'92:** Classify a chosen set of existing mutual exclusion and replica control algorithms currently in use in dis-

tributed systems using the criteria developed above.

4. **August'92 – February'93:** Develop simulation tools to evaluate the mutual exclusion and replica control algorithms. Also, develop analytical tools to solve design problems such as optimal degree of replication, placement of resources, etc. The analytical tools should also be developed to approximate the expected performance of the systems. These approximations could be used to validate the results obtained through simulations.

5. **December'92 – February'93:** Attempt to modify the existing mutual exclusion algorithms to improve their applicability to real-time applications.

6. **February'93 – April'93:** Attempt to modify the existing replica control algorithms to meet the needs of real-time applications.

7. **February'93 – July'93:** Evaluate the modified mutual exclusion and replica control algorithms in terms of the criteria chosen above, and use the tools developed above.

8. **July'93 – September'93:** Interpret the obtained results and draw conclusions regarding the desirable characteristics of mutual exclusion and replica control algorithms. Using these, identify key factors of applications and the systems that need closer scrutiny before we select/design a system for a given application. Illustrate the efficacy of this approach by developing algorithms for a chosen set of applications. Also formulate general guidelines for future developments.

9. **September'93 – December'93:** Complete writing the thesis and defend.

References

[1] T. Ae, M. Yamashita, and H. Matsumoto. A response time estimation of real-time networks. *Proceedings real-time systems symposium*, pages 198–207, Dec.87.

[2] Divyakant Agrawal and Amr El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM transactions on computer systems*, Vol 9, No. 1:1–20, Feb.91.

[3] B. S. Bacarisse and S. Bek Baydere. Reliability of replicated files in partitioned networks. *IEEE*, pages 98–101, Mar.90.

[4] Daniel Barbara and Hector Garcia-Molina. The case for controlled inconsistency in replicated data(position paper). *IEEE*, pages 35–38, Mar.90.

[5] Devesh Bhatt, Adel Ghonami, and Ranga Ramanujan. An instrumented testbed for real-time distributed systems development. *Proceedings real-time systems symposium*, pages 241–250, Dec.87.

[6] Ye-In Chang, Mukesh Singhal, and Ming T. Liu. An improved o(log n) mutual exclusion algorithm for distributed systems. *1990 International Conference on parallel processing*, pages III295–III302, 90.

[7] Ye-In Chang, Mukesh Singhal, and Ming T. Liu. A fault tolerant algorithm for distributed mutual exclusion. *9th Synposium on Reliable Distributed System*, Oct.90.

[8] W. W. Chu and C.-M. Sit. Estimating task response time with contentions for real-time distributed systems. *Proceedings real-time systems symposium*, pages 272–281, Dec.88.

[9] Bruno Ciciani, Daniel M. Dias, and Philip S. Yu. Analysis of replication in distributed database systems.

[10] L. Ciminiera and A. Valenzano. Performance analysis of acknowledgment mechanisms in token-bus networks. *Proceedings real-time systems symposium*, pages 179–185, Dec.87.

[11] Anna Hac. A distributed algorithm for performance improvement through file replication, file migration, and process migration. *IEEE transactions on software engineering*, Vol.15, No.11:1459–1470, Nov.89.

[12] Maurice Herlihy. Type-specific replication algorithms for multiprocessors. *IEEE*, pages 70–74, Mar.90.

[13] Hui-I Hsiao and David J. DeWitt. Replicated data management in the gamma database machine. *IEEE*, pages 79–84, Mar.90.

[14] James Jay Kistler. Increasing file system availability through second-class replication. *IEEE*, pages 65–69, Mar.90.

[15] Akhil Kumar. Performance analysis of a hierarchical quorum consensus algorithm for replicated objects. *IEEE*, Jul.90.

[16] Rivka Ladin. Lazy replication: Exploiting the semantics of distributed services. *IEEE*, pages 31–34, Mar.90.

[17] L. Lamport. Time, clocks and ordering of events in distributed systems. *Communications of ACM*, pages 558–565, Jul.78.

[18] Shem-Tov Levi and Ashok K. Agrawala. Real-time system design, 90.

[19] Shem-Tov Levi, Daniel Mosse, and Ashok K. Agrawala. Allocation of real-time computations under fault tolerance constraints. *Proceedings real-time systems symposium*, pages 161–171, Dec.88.

[20] Darrell D. E. Long. Analysis of replication control protocols. *IEEE*, pages 117–122, Mar.90.

[21] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in dicentralized systems. *ACM transaction on computer systems*, pages 145–159, May.85.

[22] Frank William Miller. A predictive real-time scheduling algorithm. *Department of computer science, the university of Iowa*, Aug.89.

[23] Masaaki Mizuno, Mitchell L. Neilsen, and Raghavendra Rao. A token based distributed mutual exclusion algorithm based on quorum agreements. *IEEE*, pages 361–368, Jul.91.

[24] Sape Mullender. *Distributed systems.* ACM, 89.

[25] R. D. Nelson and B. R. Iyer. Analysis of a replicated data base. *Performance Evaluation*, pages 133–148, May.85.

[26] Francis J. Prusker, Edward Pl Wobber, Wei Zhao, Krithivasan Ramamritham, and John A. Stankovic. The siphon: managing distant replicated repositories scheduling tasks with resource requirments in hard real-time systems. *IEEE proceedings of the workshop on management of replicated data IEEE transactions on software engineering*, Vol. SE-13, No. 5:44–47 564–573, Nov.90 May.87.

[27] Calton Pu, Avraham Leff, Frederick Korz, and Shu-Wie Chen. Valued redundancy. *IEEE*, pages 76–78, Mar.90.

[28] Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM transactions on computer systems*, pages 61–77, Feb.89.

[29] Beverly A. Sanders. The information structure of distributed mutual exclusion algorithms, Jun.86.

[30] Mukesh Singhal. A dynamic information-structure mutual exclusion algorithm for distributed systems. *IEEE ICDCS*, pages 70–78, 89.

[31] Mukesh Singhal. Theory and construction of optimal dynamic information-structure mutual exclusion algorithms for distributed systems. *IEEE transaction on parallel and distributed systems*, pages 70–78, 89.

[32] Mukesh Singhal. Update transport: a new technique for update synchronization in replicated database systems. *IEEE transactions on software engineering*, Vol.16, No.12:1325–1336, Dec.90.

[33] Mukesh Singhal and A. K. Agrawala. A concurrency control algorithm and its performance for replicated database systems. *IEEE*, pages 140–147, Sep.86.

[34] S. H. SON. Using replication for high performance database support in distributed real-time systems. *Proceedings real-time systems symposium*, pages 79–85, Dec.87.

[35] Jay K. Strosnider, Tom Marchok, and John Lehoczky. Advanced real-time scheduling using the ieee 802.5 token ring. *Proceedings real-time systems symposium*, pages 42–52, Dec.88.

[36] U. Sumita and O. R. Liu Sheng. Analysis of query processing in distributed database systems with fully replicated files: A hierarchical approach. *Performamce Evalluation*, pages 223–238, Aug.88.

[37] H. F. Wedde, G. S. Alijani, F. Kang, and B.-K. Kim. Melody: a distributed real-time testbed for adaptive systems. *Proceedings real-time systems symposium*, pages 112–119, Dec.88.

[38] C. M. Woodside and D. W. Craig. Local non-preemptive scheduling policies for hard real-time distributed systems. *Proceedings real-time systems symposium*, pages 12–16, Dec.87.

[39] Wei Zhao, Krithivasan Ramamritham, and John A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE transactions on computers*, Vol. C-36, No. 8:949–960, Aug.87.